Thomas Bartz-Beielstein
Marco Chiarandini
Luís Paquete · Mike Preuss (Eds.)

Experimental Methods for the Analysis of Optimization Algorithms

Springer

Experimental Methods for the Analysis of Optimization Algorithms

Thomas Bartz-Beielstein · Marco Chiarandini ·
Luís Paquete · Mike Preuss
Editors

Experimental Methods for the Analysis of Optimization Algorithms

Springer

Editors

Prof. Dr. Thomas Bartz-Beielstein
Cologne University of Applied Sciences
Institute of Computer Science
Faculty of Computer and
Engineering Science
Campus Gummersbach
Steinmüllerallee 1
51643 Gummersbach
Germany
thomas.bartz-beielstein@fh-koeln.de

Dr. Marco Chiarandini
University of Southern Denmark
Department of Mathematics and
Computer Science
Campusvej 55
5230 Odense
Denmark
marco@imada.sdu.dk

Dr. rer. nat. Luís Paquete
University of Coimbra
CISUC
Department of Informatics Engineering
Pólo II
3030-290 Coimbra
Portugal
paquete@dei.uc.pt

Mike Preuss
TU Dortmund
Department of Computer Science
Algorithm Engineering
Otto-Hahn-Str. 14
44227 Dortmund
Germany
mike.preuss@tu-dortmund.de

ISBN 978-3-642-02537-2 e-ISBN 978-3-642-02538-9
DOI 10.1007/978-3-642-02538-9
Springer Heidelberg Dordrecht London New York

ACM Codes: G.1, G.3, F.2, I.2

Cover design: KuenkelLopka GmbH

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Foreword

This book belongs on the shelf of anyone interested in carrying out experimental research on algorithms and heuristics for optimization problems.

The editors have brought together expertise from diverse sources to address methodological issues arising in this field. The presentation is wide-ranging, containing "big picture" discussions as well as more focused treatment of specific statistical techniques and their application. The emphasis throughout is on careful process and scientific rigor; the discussion is illuminated with many case studies, small tutorials, and references to the literature on optimization.

Don't keep this book on the shelf: read it, and apply the techniques and tools contained herein to your own algorithmic research project. Your experiments will become more efficient and more trustworthy, and your experimental data will lead to clearer and deeper insights about performance.

Amherst, Massachusetts, February 2010 *Catherine C. McGeoch*

Foreword

Once upon a time, more exactly nearly half a century ago, when the first cybernetic machines, henceforth called computers, became available to academic institutions, a few people seemed to have waited for their iterative power to perform otherwise boring procedures like solving sets of linear equations, etc. Among other ideas to make use of their tireless working through loops of instructions was the simulation of organic evolution, the main subroutines of which are mutation, recombination, and natural selection. It was only a small step to imagine that by means of the same principles the design of technical devices, managerial tasks and other systems could be stepwise improved, if not even optimized. Competing methods from numerical mathematics were known, of course, but also their limitations to linear and quadratic dependencies between decision variables and objectives. In so-called black box situations where much more complex dependencies prevail and nonlinear constraints, stochastic disturbances, and the like hamper the search for optima, using evolutionary variation and evaluation processes showed up their capacities. Evolutionary algorithms thus were born during the 1960s, and they have matured ever since to a powerful and broadly accepted tool within many disciplines. Together with two other modern streams, artificial *neural networks* (NN) and *fuzzy systems* (FS), they have been subsumed into the so-called *computational intelligence* (CI) field, at least since 1994, when the first world congress on CI took place with its three subbranches NN, FS, and EC (evolutionary computation).

In the beginning of EC one had to be happy if one could rerun a numerical experiment a few times, for example with different seeds of the pseudo random generator or different start positions in the search space. Gathering a whole set of statistical data was unimaginable then, so that many open questions remained about the performance of the algorithms. What are those questions? It's not only an average value and its variance and skewness, or the best result out of a few runs that are interesting. One wants to know whether the ultimate result, i.e. using the algorithm specific stopping criterion, is always the same, whether and how it depends on the random numbers and starting points used. Further it may happen or not that fatal execution errors occur, like division by zero or extracting the square root of a negative number, or that the stopping criterion does not work properly – even if the optimum was found exactly. And what exactly does it mean to talk of four or eight precise digits or even more (which may depend on the hardware and on the system software handling real numbers, their mantissae and exponents)? Introducing common stopping criteria to compare different methods can easily deliver controversial results depending on the slopes of best (or average) intermediate results over the number of objective function values. Such slopes can not only have one crossing, but even two or more. Then the result depends heavily upon the number of admitted iterations, one method being quicker at the beginning while finally delivering mediocre final results, or the other way round, or even more complicated.

If you are interested in such questions, THIS is the book to look into. Here you will find even more aspects that are treated scientifically by the experts in that exciting domain offering their up-to-date know-how and even leading into philosophical domains.

Dortmund, February 2010 *Hans-Paul Schwefel*

Preface

Optimization algorithms are used to solve problems that arise in relevant research and application areas such as operations research, computer science, and engineering. During recent decades the experimental approach has been recognized and accepted in the analysis of these algorithms and a considerable body of research has been devoted to the development and establishment of an adequate scientific methodology for pursuing this kind of analysis. Statistical tools have become more and more popular. This book is written for researchers and practitioners of operations research and computer science who wish to improve the experimental assessment of their optimization algorithms with the final goal of improving their design. It collects prominent methodological works on different scenarios of experimental analysis.

The book consists of an introduction and 4 chapters written by the editors plus 11 chapters (including an appendix) written by *invited* contributors. All together the project involved 30 authors of 16 world-wide academic institutions.

The first part of the book lays the basis giving an all-round view of the issues involved in the experimental analysis of algorithms. The second part treats the characterization by means of statistical distributions of the algorithm performance in terms of solution quality, run-time, and other measures. The third part collects advanced methods from experimental design for configuring and tuning algorithms on a specific class of instances with the goal of using the least amount of experimentation and attaining sound conclusions. Several chapters are enriched with case studies.

Acknowledgments

We are indebted to many people, especially to the contributors of this volume.

The book is the result of an inspiring and fruitful cooperation among the editors. First contacts were established during the Metaheuristic International Conference in Vienna in 2005 — followed by research visits and meetings during conferences.

We realized that problems that we had been trying to solve are of interest for a broader audience and not restricted to our own research areas (computational intelligence, operations research, mathematical optimization). Certainly, a milestone in the pathway to this book was the Workshop on Empirical Methods for the Analysis of Algorithms held in Reykjavik in 2006. We are thankful to all the participants of that workshop, many of whom became authors in this book and contributed to define its scope. Among them, a special mention goes to Catherine McGeoch whose perspective in the experimental analysis of algorithms highly influenced our views.

On an individual basis, Bartz-Beielstein gratefully acknowledges the support of colleagues at Cologne University of Applied Sciences: Wolfgang Konen for the inspiring intellectual exchange and Michael Bongards for providing valuable input for real-world applications. Some of Thomas Bartz-Beielstein's work was conducted under a Bundesministerium für Forschung und Bildung (BMBF) grant (FIWA, AIF FKZ 17N2309) and under a Cologne University of Applied Sciences grant (COSA). Thomas Bartz-Beielstein would like to thank Deborah Mayo for exchanging ideas about error and inference. Many thanks go to Jack P.C. Kleijnen, Ekaterina Vladislavleva, and Dick den Hertog for invitations to the wonderful mini-symposia at Tilburg University. Thomas Bartz-Beielstein and Mike Preuss are grateful to Hans-Paul Schwefel and Günter Rudolph for detailed comments, valuable suggestions and inspiring advice over many years. We thank Gwenn Volkert, Jürgen Branke, Mark Wineberg, and Steffen Christensen for promoting the idea of experimental research during the leading conferences in evolutionary computation. Ruxandra and Catalin Stoean also deserve thanks for the open-mindedness to jointly investigate every possible algorithmic variation. This also applies to the colleagues at RWTH Aachen, Christoph Kausch, Frank Henrich and Claude Bouvy, who added valuable insight into industrial optimization problems in work conducted together under DFG project grant 252441. Marco Chiarandini is thankful to Yuri Goegebeur for his advice on statistical issues during the reviewing process of the chapters. Luís Paquete is grateful to Carlos Fonseca and Thomas Stützle for the encouraging advice and stimulating exchange of ideas about the main topic of this book. He also acknowledges his colleagues at University of Coimbra, mainly Ernesto Costa and Alberto Moraglio, for the enlightening discussions on design and analysis of stochastic algorithms for optimization.

Finally, we all would like to thank Ronan Nugent of Springer, who provided great support during all production stages of this book.

Gummersbach, Odense, Coimbra, Dortmund,
February 2010

Thomas Bartz-Beielstein
Marco Chiarandini
Luís Paquete
Mike Preuss

Contents

List of Contributors

Prasanna Balaprakash
IRIDIA, CoDE, Université Libre de Bruxelles,
Brussels, Belgium
e-mail: pbalapra@ulb.ac.be

Thomas Bartz-Beielstein
Institute of Computer Science, Cologne University of Applied Sciences
Gummersbach, Germany
e-mail: thomas.bartz-beielstein@fh-koeln.de

Dario Basso
Department of Statistics, University of Padua
Padua, Italy
e-mail: dario@stat.unipd.it

Mauro Birattari
IRIDIA, CoDE, Université Libre de Bruxelles
Brussels, Belgium
e-mail: mbiro@ulb.ac.be

Marco Chiarandini
Department of Mathematics and Computer Science, University of Southern Denmark
Odense, Denmark
e-mail: marco@imada.sdu.dk

Markus Chimani
Algorithm Engineering, TU Dortmund
Dortmund, Germany
e-mail: markus.chimani@tu-dortmund.de

Agoston E. Eiben
Vrije Universiteit
Amsterdam, The Netherlands
e-mail: gusz@cs.vu.nl

Viviane Grunert da Fonseca
CEG-IST – Centre for Management Studies, Instituto Superior Técnico
Lisbon, Portugal
INUAF – Instituto Superior D. Afonso III
Loulé, Portugal
e-mail: viviane.grunert@vodafone.pt

Carlos M. Fonseca
CEG-IST – Centre for Management Studies, Instituto Superior Técnico
Lisbon, Portugal
Department of Electronic Engineering and Informatics, Faculty of Science and Technology, Universidade do Algarve
Faro, Portugal
e-mail: cmfonsec@ualg.pt

Matteo Gagliolo
IRIDIA, CoDE, Université Libre de Bruxelles
Brussels, Belgium
Faculty of Informatics, University of Lugano
Lugano, Switzerland
e-mail: mgagliolo@iridia.ulb.ac.be

Yuri Goegebeur
Department of Mathematics and Computer Science, University of Southern Denmark
Odense, Denmark
Research Group Quantitative Psychology and Individual Differences, K.U.Leuven
Leuven, Belgium
e-mail: yuri.goegebeur@stat.sdu.dk

Nicholas G. Hall
Department of Management Sciences, The Ohio State University
Columbus, Ohio, USA
e-mail: hall_33@fisher.osu.edu

Holger H. Hoos
Department of Computer Science, University of British Columbia
Vancouver, Canada
e-mail: hoos@cs.ubc.ca

Jürg Hüsler
Department of Mathematical Statistics, University of Bern
Bern, Switzerland
e-mail: juerg.huesler@stat.unibe.ch

Frank Hutter
Department of Computer Science, University of British Columbia
Vancouver, Canada
e-mail: hutter@cs.ubc.ca

Jack P.C. Kleijnen
Department of Information Management / CentER, Tilburg University
Tilburg, The Netherlands
e-mail: kleijnen@uvt.nl

Karsten Klein
Algorithm Engineering, TU Dortmund
Dortmund, Germany
e-mail: karsten.klein@tu-dortmund.de

Daniel Kudenko
Department of Computer Science, The University of York
York, UK
e-mail: Kudenko@cs.york.ac.uk

Christian Lasarczyk
Algorithm Engineering, TU Dortmund
Dortmund, Germany
e-mail: Christian.Lasarczyk@udo.edu

Catherine Legrand
Institut de statistique, Université catholique de Louvain
Louvain-la-Neuve, Belgium
e-mail: catherine.legrand@uclouvain.be

Kevin Leyton-Brown
Department of Computer Science, University of British Columbia
Vancouver, Canada
e-mail: kevinlb@cs.ubc.ca

Manuel López-Ibáñez
IRIDIA, CoDE, Université Libre de Bruxelles
Brussels, Belgium
e-mail: manuel.lopez-ibanez@ulb.ac.be

Kevin P. Murphy
Department of Computer Science, University of British Columbia
Vancouver, Canada
e-mail: murphyk@cs.ubc.ca

Luís Paquete
CISUC, Department of Informatics Engineering, University of Coimbra
Coimbra, Portugal
e-mail: paquete@dei.uc.pt

Marc E. Posner
Department of Integrated Systems Engineering, The Ohio State University
Columbus, Ohio, USA
e-mail: posner.1@osu.edu

Mike Preuss
Algorithm Engineering, TU Dortmund
Dortmund, Germany
e-mail: mike.preuss@tu-dortmund.de

Enda Ridge
Forensic Technology and Discovery Services, Ernst and Young
London, UK
e-mail: enda.ridge@gmail.com

Selmar K. Smit
Vrije Universiteit
Amsterdam, The Netherlands
e-mail: sksmit@cs.vu.nl

Thomas Stützle
IRIDIA, CoDE, Université Libre de Bruxelles
Brussels, Belgium
e-mail: stuetzle@ulb.ac.be

Zhi Yuan
IRIDIA, CoDE, Université Libre de Bruxelles
Brussels, Belgium
e-mail: zyuan@ulb.ac.be

Chapter 1
Introduction

Thomas Bartz-Beielstein, Marco Chiarandini, Luís Paquete, and Mike Preuss

Abstract Theory and experiments are complementary ways to analyze optimization algorithms. Experiments can also live a life of their own and produce learning without need to follow or test a theory. Yet, in order to make conclusions based on experiments trustworthy, reliable, and objective a systematic methodology is needed. In the natural sciences, this methodology relies on the mathematical framework of statistics. This book collects the results of recent research that focused on the application of statistical principles to the specific task of analyzing optimization algorithms.

1.1 Optimization Algorithms

Optimization problems arise in many contexts of operations research, computer science, engineering and other research and application areas. Designing constructive mathematical abstractions, called *algorithms*, is a common approach to solve these problems. Algorithms that are used to solve optimization problems can be essentially divided into two different types: (i) exact algorithms and (ii) heuristic algorithms.

Thomas Bartz-Beielstein
Institute of Computer Science, Cologne University of Applied Sciences, 51643 Gummersbach, Germany, e-mail: thomas.bartz-beielstein@fh-koeln.de

Marco Chiarandini
Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark, e-mail: marco@imada.sdu.dk

Luís Paquete
CISUC, Department of Informatics Engineering, University of Coimbra, Portugal, e-mail: paquete@dei.uc.pt

Mike Preuss
Algorithm Engineering, TU Dortmund, Germany, e-mail: mike.preuss@tu-dortmund.de

Exact algorithms compute provably optimal solutions. They can be general-purpose algorithms, such as simplex and interior-point methods for mathematical programming formulations, and complete search algorithms, such as backtracking and graph-search algorithms. They may also be more problem specific, such as the Dijkstra's algorithm for finding the shortest path in a graph, the Ford–Fulkerson's algorithm to maximize the flow in a network, or the dynamic programming algorithm for knapsack problems.

Heuristic algorithms (here shortened to *heuristics*) are algorithms for which it is not possible to prove that the optimum will be reached in finite time. The term *heuristic* is of Greek origin, meaning "serving to find out or discover." Hence, a heuristic can be considered as a useful shortcut, an approximation, or a rule of thumb for guiding search. In practice, heuristics can achieve good performance both in terms of runtimes and in terms of solution quality. They are frequently applied when time constraints are too tight for exact algorithms. Their use is predominant in relevant application areas of combinatorial optimization, such as routing (Golden et al. 2008), scheduling (Van Hentenryck and Michel 2005), and timetabling Burke et al. (2008). Typical examples of heuristics are greedy algorithms that construct a solution by series of choices with the best local return. A particular class of heuristics is made by *approximation algorithms*. They are understood as polynomial time routines with a guaranteed level of performance in terms of solution quality. Approximation algorithms are designed for many hard optimization problems, such as for the set covering, the bin packing and several machine scheduling problems.

Finally, there is the well-known class of *metaheuristics*. Metaheuristics are general algorithmic frameworks that can be applied to a wide spectrum of problems for producing heuristics. Typically, applying a metaheuristic is a rather simple process that requires us only the design of problem specific components to be used within the framework. Often, the guiding principle of a metaheuristic is inspired by some natural phenomena, such as in the case of simulated annealing (Kirkpatrick et al. 1983), evolutionary algorithms (Schwefel 1995), and ant algorithms (Dorigo and Stützle 2004). In continuous optimization, it sometimes sufficient to define a quality function that, for two possible solutions, states whether they are equally good or indicates which one is better, a situation known as *black-box optimization*.

Optimization algorithms are most commonly randomized. The reasons for randomization are various: possibility of gains from reruns, the adversary argument, structural simplicity for comparable average performance, speed up, avoiding loops in the search, and so on (Motwani and Raghavan 1995). Exact algorithms, when randomized, still return an optimal solution but have random runtime. Randomized heuristics have instead both solution quality and runtime random.

1.2 Analysis of Algorithms

1.2.1 Theoretical Analysis

In computer science, algorithms and machines to run them are formally described. In particular, the Turing machine is the formal model within which any computable algorithm can be analyzed. This led to the idea that there is no need to understand details of the hardware, because the high-level description of the algorithms will work the same on any physical system, provided that it is stable enough to carry out the programs. This is true if algorithms can be investigated formally and if a general and, to a certain extent, incomplete description of the algorithm behavior is sufficient.

The themes of theoretical work on algorithms, here intended as well-substantiated explanations derived by mathematical logic, are broad. Solving problems in the first place can be seen as theoretical work comprising intuition, deduction and proof. Examples are finding algorithms that solve complicated problems without using bruteforce and complete enumeration, or reductions between problems arising in NP-completeness theory. Restricting ourselves to the analysis of randomized algorithms we can distinguish different kinds of theoretical analysis, without aiming at being exhaustive. Worst-case and average-case are the classical analyses that can be carried out both on solution quality approximation and on runtime. These include probabilistic analysis for the expected runtime. Domination analysis counts the number of solutions that are dominated by the solutions returned by an algorithm on the instance where it performs worst. Convergence analysis studies the behavior of the algorithm when runtime increases to infinite. Other analyses may focus on proving the usefulness of some algorithmic components or the invariance of the algorithm with respect to some trivial transformation of the input data.

1.2.2 Experimental Analysis

Computers are human artifacts constructed to execute algorithms and solve practical problems. They make it possible to study computation as a natural occurrence, not only by formal analysis, but also by experimentation. Algorithms that would be too complex to study analytically become accessible via empirical inquiry. In addition, experimentation can provide more accurate predictions of the behavior of an algorithm in practice.

In experiments, the object of analysis are not abstract algorithms but application and simulation programs running in a particular computing environment and solving problems for real input instances (McGeoch 1996). Simulation programs differ from application programs that are used in practice only in that they provide the scientist with complete control of the experimental environment. Implementation details may have a certain impact on the performance of the programs that are derived from the algorithms under study. These details are omitted for simplicity in a mathematical analysis and unfortunately, they are often omitted also in descriptions of experimental work. McGeoch (1996) distinguishes different *levels of description* for the algorithms analyzed. For instance, a metaheuristic is "a minimally instantiated algorithm" that contains few implementation details and that is probably better defined as an algorithm template. A more instantiated algorithm may incorporate basic implementation strategies (such as whether to use stacks or queues for a given data structure); an even more instantiated algorithm may specify programming tricks that give constant-factor speeds. A highly instantiated algorithm might describe specific details for a particular programming language or computer architecture. We deem it important being aware of these distinctions. Accordingly, a certain degree of expertise in the field is required to know the level of description of an algorithm to which the results of a program can be abstracted. In other words, the exploration of the conditions that influence the study of an algorithm at the intended level of description is part of the work of the experimenter. His skills and ingenuity enable him to remove unwanted factors and to make experiments relatively easy to realize and to reproduce.

Theoretical analyses provide asymptotic results or assume special types of input data that are different from those in which the problem usually occurs in a real-life context. Moreover, results from theory rely on a standard machine model for computation that is far from modern computers. Hence, they lead to predictions that are too vague for practical purposes. Additionally, theoretical analyses have to fix concrete problems or at least assume that the problems, treated strictly, adhere to some requirements, such as, for example, the objective function being convex. As the algorithms applied in practice are often quite complex, these also have to be simplified to be theoretically tractable. The insight gained by a theoretical analysis is still valuable but may not be applicable to any practical use case. Using an experimental approach, we are no longer bound to design simple algorithms that are amenable to mathematical analysis. We can focus on complex algorithms whose theoretical analysis would be beyond human ability, but whose performance better satisfies practical needs. Similarly, this freedom can be extended to problems that are more complex and closer to real-life situations. Experiments can inspire theoreticians and be used by them to disprove conjectures, gain new insights for their work, or suggest new directions for theoretical analysis.

Hence, experimentation on algorithm implementations on up-to-date computers is relevant in practice and definitely needed if we wish to achieve more accurate predictions about their performance and robustness. The following section details how experiments can benefit from theory and vice versa.

1.3 Bridging the Gap Between Theoretical and Empirical Analysis

As is widely understood, science is a systematic search for knowledge about the world, resulting in a prescriptive practice and prediction of a type of outcome. In many scientific disciplines, e.g., physics or chemistry, there is nowadays agreement that theories and experiments are equally relevant. Researchers in theoretical physics collaborate with experimentalists to submit their theses to experimental testing and, viceversa, experiments inspire new approaches.

The situation in the field of computing is much less well understood. Since algorithms are mathematical abstractions there is a considerable area in the field of computing that maintains a purely formal approach, quite similar to in mathematics and the formal sciences. Some researchers in this area hold that a theoretical proof of the behavior of an (often simplified) algorithm is of primary interest. However, as mentioned above, to be of any use algorithms must be written into a programming language and run on a computer. This necessarily transforms a mathematical abstraction into a real-world matter that calls for a natural-science approach. Unfortunately, in the field of computing, contrary to in physics, theoretical results are seldom subjected to empirical test or, as would be often more correctly stated in this context, they are seldom submitted to *refinement*.

We observe instead a different approach to experimentation that is widespread in other areas of computing, such as heuristics and evolutionary computation. In this approach there is only a weak link with theory, in the sense that there is not a deductively inferred theory to put to test but rather intuitive ideas on what would make easier to solve a certain problem. In fact, a considerable number of scientists in this area of computing implement these ideas in algorithms and try them on empirical data without any prior analytical analysis. They observe the results and refine their design. They proceed inductively, from experimental data. What we observe is a learning process from experimentation rather than a theoretical development.

Our impression is then that, in the field of computing, theorists and experimentalists still live in two separate worlds. A serious attempt to bridge the gap between these worlds is known under the name of *algorithm engineering*; see also the contribution of Chimani and Klein in Chap. 6 of this book. The view of its proponents is to test empirically theories about algorithms. The idea put forward is that theoretical and experimental work can inspire each other (as was already envisioned by Galileo) in an iterative process, which they call the engineering cycle, and which has the final goal of improving the design of the algorithms.

A peculiarity of computer science, with respect to other natural sciences, is that theories are always true because they concern mathematical abstractions and are obtained by mathematical reasoning. Nevertheless, they are often imprecise because implementation aspects such as programming language and computer architecture are not taken into consideration. Theories are therefore insufficient for good prediction. Hence, well-conducted experiments enable us to *refine* theories, when any are available, and to *learn*. Experiments can positively identify previously unknown effects and are complementary to theories.

The description of the previous paragraphs is clearly specific of the field of computing. However, we can find in the contemporary philosophy of natural sciences a more general debate on the role of experiments and their importance (Chalmers 1999). In particular, two prominent philosophers, Hacking (2001) and Mayo (1996), give a sustained treatment of experiments as independent of theory, interacting with it, as well as with invention and technology, in numerous ways and with different relationships in different sciences at different stages of development. Experiments acquire a new stance and can be used as a starting point from which evidence emerges. This new way of looking at experiments is known under the term *new experimentalism* (Ackermann 1989). Mayo proposes learning about the world by actively probing, manipulating, and simulating patterns of error in experiments. Central to Mayo's approach is the concept of severity that is directly connected to learning from error. The experimenter learns that an error is absent when a procedure of inquiry (which may be based on several tests) that has a very high probability of detecting an error if it exists nevertheless detects no error. A key goal of the new experimentalism is to reinterpret classical statistical tests as tools for obtaining knowledge. Mayo (1983) presents the formal statistical framework. Bartz-Beielstein (2008) transfers this framework to experimental approaches in computer science (see also Chap. 2 of this book).

Summarizing, there are at least two interesting ongoing processes in the field of computing that try to bridge the gap between theory and experiment: (i) algorithm engineering, that starts with theories and refines them and (ii) the new experimentalism, that starts from experiments without any high-level theory and lets evidence arise as a matter of fact. The former can be classified as a deductive approach, whereas the latter is related to inductive approaches and reliabilism. Note, and this is a relevant observation, *statistics* plays a central role in both approaches.

With respect to the situation presented, our intention in this book is threefold. First, we wish to give importance to experiments and qualify them as complementary to theories, as ways to test and refine them, improve them, and make them more meaningful and useful in practice. Second, we intend to recognize the existence of a scientific process in the field of computing that consists of applying statistical testing and learning from error. This approach is interpreted within the position of Hacking and Mayo. Accordingly, we regard the learning that we achieve from experiments as valuable and trustworthy. Finally, because it is a common premise for achieving the two previous aims, we wish to contribute to make the analysis of experiments in the field of computing more rigorous, objective and reproducible, hence similar to what is seen in other natural sciences such as physics, biology, and chemistry. As we explain in the next section, this goes necessarily through the adoption of the statistical framework.

1.4 The Need for Statistics

So far we have argued for the relevance of experimentation in computing. But how should experimentation be carried out in order to gain a scientific stance? What are the pitfalls we would like to avoid? Prominent authors, like Hooker (1996) and Johnson (2002), have already replied to this question, discussing extensively issues related to the experimental study of algorithms. These articles are good starting points for any experimenter in the field.

In this book, we further advocate the use of statistics as a systematic way of analysis. Statistics offers a well-developed and accepted mathematical framework to the analysis of experimental data. It suggests ways to look at data to discover relevant patterns and techniques to plan experiments intelligently and separate effects. It also provides quantitative assessment of the inference in which the scientist is interested.

In the case of randomized algorithms, we need to protect the researcher from falsely concluding about the presence of an effect caused by the algorithm when there is none. The direction of the observed differences might be simply due to chance and explained by sampling variance. Nevertheless, if algorithms differ in even the smallest component then they should, ultimately, lead to different results. Thus, it is important to consider the entity of the true differences: they might be so small that we are in fact scientifically indifferent to them. On the other side, we might want to be protected against erroneously concluding that no such difference exists when one does. These concepts are made formal and quantitative in statistics by fixing the level of significance and the statistical power of the chosen test procedure. In this way, we bound within an agreed limit the chances of making a certain kind of mistake if the test procedure were to be repeated a large number of times.

In computing we may distinguish different scenarios where statistics is needed. On the one side, we may wish to draw conclusions on the basis of small samples because it is computationally costly to run experiments. This situation occurs in many real-world optimization scenarios, e.g., in engineering design where one single function evaluation might be the result of a complex simulation. In this case statistics helps us to avoid premature conclusions that are not evident from the data and to design the experiments in the most effective way. On the other side, computational cost might not be a problem, in which case we could run as many experiments as we wish but might want to have a unique decision at the end. This situation might occur while optimizing artificial test functions that were defined to understand the behavior of algorithms. Still here, statistics helps us to maximize the space of tested algorithmic configurations, telling us when the data collected are sufficient to determine a difference and suggesting the direction for new experiments. Alternatively, it can provide confidence intervals around the estimated effects, thus increasing the level and precision of the information collected. It is possible, of course, to find situations that stay half-way between the two described here. For example, it might be possible to decide to stop the experiments before a unique answer has been found. This may be done when a time deadline is reached or when enough statistical power has been collected to detect a minimal effect of scientific interest and this has not yet become evident in the data collected.

One of the criticisms of experimental research in computing is that the results produced are less trustworthy than theoretical ones because they are contextual, architecture dependent and hence less general and unreproducible. Indeed, we agree, often computational experiments found in the literature are not severe enough and many claims remain invalidated. But our point of view is that this criticism should not hinder experimentation, rather, through its awareness, it should rise its quality and lead to a more sophisticated methodology. In our vision computational experiments are more complex than it is believed and require field expertise to deal with the various sources of variance. They go beyond running a few experiments on a single machine and collecting results in a table. They entail summarizing, visualizing, and testing data with principled methods of analysis taken from statistics to distinguish the effects of different sources. The ultimate goal is a useful and generalizable description of algorithm behavior. We will achieve these goals by borrowing from well-established rules on experimentation from other fields and developing new specific tools. The final goal is the establishment of standards for experimental research in computing. The methods arising from considerations of this kind may then be of advantage not only to the scientist but also to the practitioner and designer, providing them with the tools for correct and fast decision making.

1.5 Book Contents

The analysis of optimization algorithms focuses primarily on two measures of performance *solution cost* and *runtime*. From a statistical point of view these are random variables and in experimentation we base our description on finite-sized sampled data.

In statistics we distinguish three areas of data analysis: descriptive statistics, design of experiments, and inferential statistics. *Descriptive statistics* deals with the summary and graphical representation of results. *Design of experiments* provides a systematic framework for the collection and evaluation of data. Finally, *inferential statistics* supplies a probabilistic measure of events on the basis of mathematical derivations from the empirical data.

In this book we collect important techniques from all these areas that may contribute to attaining an empirical assessment in different scenarios of analysis. Given our aim of a book oriented to practice, every chapter contains both an explanation of the techniques and their example application to case studies.

We have organized the chapters into three parts, where similar questions are addressed. In Part I, the focus is on the object of analysis, the algorithms, and the problem instances. In Part II, the focus is on the characterization of the probability distribution of the random variable chosen to measure algorithm performance. In Part III, we group all applications of experimental design techniques for modeling the relationships between algorithm parameters, instance features, and algorithm performance. The goal of all these methods is the separation of effects and the comparison of average performance. The differences in size of these parts reflects the attention devoted to the different scenarios in the literature.

Most of the chapters have a more or less involved statistical content. The reader who is not very confident with this subject can find in the Appendix a very precise introduction to the theory of inferential statistics by Dario Basso.

The first part starts with "The Future of Experimental Research" by Thomas Bartz-Beielstein and Mike Preuss, who examine the scientific method and its use in computer science. They discuss philosophical foundations, the goal of experimentation, the use of statistics, and the interpretation of results. While recognizing the need for statistics, the authors warn the reader that statistics is not all. Other issues such as the scientific meaning of a result must also be taken into consideration. The view put forth is that experiments can be used to discover "theories" rather than only to test theory. This view is consistent with the current of new experimentalism in the contemporary philosophy of science.

The following chapter, "Design and Analysis of Computational Experiments: Overview" by Jack P.C. Kleijnen, introduces the reader to experimental design and modeling by means of classical and modern regression techniques. It includes both methods that were successfully applied to the tuning of algorithms and that we will encounter in the third part of the book as well as methods that have not yet been applied and that represent interesting potential for future research.

The next chapter, "The Generation of Experimental Data for Computational Testing in Optimization" by Nicholas G. Hall and Marc E. Posner, considers issues that arise when generating random instances for testing algorithms on optimization problems. A protocol for a generation scheme is proposed and a set of generation principles is analyzed through a review of the literature of generation schemes for testing optimization algorithms in different application areas. A wealth of pointers for details on problem-specific issues is provided.

The chapter "The Attainment-Function Approach to Stochastic Multiobjective Optimizer Assessment and Comparison" by Viviane Grunert da Fonseca and Carlos M. Fonseca enlarges the object of study to multiobjective optimization. In this case, solution quality is no longer expressed by a scalar value that becomes a random variable when algorithms are randomized, but by a vector of values that becomes a random set of points in the objective space in the case of randomization. The attainment-function proposed to characterize a set of Pareto (approximate) optimal solutions is derived from random set theory.

In the last chapter of this first part, "Algorithm Engineering: Concepts and Practice," Markus Chimani and Karsten Klein give a wide introduction to experimental algorithmics and algorithm engineering. They review several issues related to the objects of the experiments of this book and provide many interesting links to follow. The chapter has the intent to bridge a gap between two communities, the algorithmic and the metaheuristic, that continue to have different venues but that could profit from more interaction.

The second part describes the characterization in terms of statistical distributions of algorithm performance, represented by solution quality or runtime.

In “Algorithm Survival Analysis,” the authors, Matteo Gagliolo and Catherine Legrand, illustrate the theoretical background of survival analysis methods applied to model runtime distributions of algorithms for solving decision problems. The method presented has a potential impact not only on the analysis but also on the improvement of the algorithms themselves, as it provides indications for restart strategies and for algorithm portfolio selection.

Jürg Hüsler, in his chapter “On Applications of Extreme Value Theory in Optimization,” models the tails of the distributions of solution quality returned by a randomized algorithm for continuous optimization. He uses distributions from extreme value theory and reports a formal proof that support their use with algorithms.

Manuel López-Ibáñez, Luís Paquete, and Thomas Stützle look at the characterization of algorithm performance in terms of Pareto-optimality in multiobjective optimization. Their chapter “Exploratory Analysis of Stochastic Local Search Algorithms in Biobjective Optimization” builds on the work of Viviane Grunert da Fonseca and C.M. Fonseca and presents a graphical representation of the empirical attainment function. This is a novel tool for exploratory data analysis in the context of biobjective optimization that makes it possible to investigate and compare by visual inspection different algorithm behaviors.

The third part collects advanced methods, mainly from experimental design, for the problem of configuring and tuning algorithms on a specific class of instances.

The chapter “Mixed Models for the Analysis of Optimization Algorithms” by Marco Chiarandini and Yuri Goegebeur gives a detailed introduction to linear statistical models for the typical scenarios of optimization algorithm studies. The goal is separating the effects of different factors in an aggregate analysis. It differentiates algorithmic components and instance features, while maintaining them under the same framework of analysis.

Enda Ridge and Daniel Kudenko in “Tuning an Algorithm Using Design of Experiments” suggest the use of more advanced experimental designs for the goal of screening the relevance of parameters with the least amount of experimentation. Successively, the determination of the best setting for the parameters left is solved by the use of response surface methods in which a response surface is modeled in the space of parameters.

“Using Entropy for Parameter Analysis of Evolutionary Algorithms,” by Selmar K. Smit and Agoston E. Eiben, proposes an alternative method to estimate the relevance of parameters with respect to screening methods from statistics. The method consists of an evolutionary algorithm (REVAC) that samples the space of parameters and collects the values of entropy, a measure of information, at different levels of performance measure. A detailed and well-explained case study is conducted, showing the application of this method to the analysis of evolutionary algorithms obtained by the combination of different choices for their parameters.

Selecting the best configuration out of a set of candidates is the topic of the next three chapters.

In “`F-Race` and Iterated `F-Race`: an Overview,” Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle present a simple but very effective approach with roots in machine learning. `F-Race` implements a sequential testing method in which candidates are discarded as soon as statistical evidence arises against them. The procedure presented is an extension of the original F-race in which new candidates can be sampled from the space of parameters on the basis of current results. The focus is on different methods for generating new design points on the basis of current results.

The last two chapters look also at sequential testing methods but they focus on the statistical models used for fitting the results and thus obtain information about where to move in the design space. The chapter "The Sequential Parameter Optimization Toolbox," written by Thomas Bartz-Beielstein, Christian Lasarczyk and Mike Preuss, exemplifies the SPO framework introduced in Chap. 2. The sequential parameter optimization toolbox SPOT is introduced as a freely available tool to improve algorithm performance. It is the result of a combination of principles from design of experiments applied to the problem of tuning the parameters of algorithms. However, the main benefit of SPOT is prevention from worse algorithm parameter configurations—and not to determine the overall-best parameter set. The example is explained using the free statistical environment R.

Finally, Frank Hutter, Thomas Bartz-Beielstein, Holger H. Hoos, Kevin Leyton-Brown, and Kevin P. Murphy in "Sequential Model-Based Parameter Optimization: An Experimental Investigation of Automated and Interactive Approaches" study two modeling methods from the literature based on Gaussian process models. These are sequential parameter optimization (SPO), which is also discussed in Chap. 14, and sequential Kriging optimization (SKO). Key design decisions within the SPO paradigm are considered. Results from these experimental studies lead to a new version of SPO, dubbed SPO$^+$. The authors compare automated parameter tuning approaches to an interactive, manual process that makes use of classical regression techniques. This interactive approach is particularly useful when only a relatively small number of parameter configurations can be evaluated. It can help human experts to gain insights into the parameter response of a given algorithm and to identify reasonable parameter settings.

The scenarios outlined above are just some of the experimental studies that can be conducted in the study of algorithms for optimization. The description centered necessarily on the content of the book. Performance measures are by no means restricted to runtime and solution quality. Other indicators may be relevant in different contexts, see, for example, McGeoch (1996), Chap. 6 in Birattari (2005) or Bartz-Beielstein (2006).

Moreover, several chapters in this book, rather than being conclusive and giving standards, outline the need for further research in this area. Correct experimental analysis of algorithms is apparently a less easy task than it may seem. The contribution of this book in this sense is to bring the reader to the cutting edge of what the available methods can do and where they might fall short.

These and other issues deserve attention and will certainly continue to attract interdisciplinary research. We are grateful to the authors of the chapters for their outstanding contributions. We hope that this book will be pleasant read and that it will contribute to improve the assessment of optimization algorithms and consequently the solution of problems arising in practice.

References

Ackermann R (1989) The new experimentalism. British Journal for the Philosophy of Science 40:185–190

Bartz-Beielstein T (2006) Experimental Research in Evolutionary Computation—The New Experimentalism. Natural Computing Series, Springer, Berlin, Heidelberg, New York

Bartz-Beielstein T (2008) How experimental algorithmics can benefit from Mayo's extensions to Neyman-Pearson theory of testing. Synthese 163(3):385–396, DOI 10.1007/s11229-007-9297-z

Birattari M (2005) Tuning Metaheuristics. Springer, Berlin, Heidelberg, New York

Burke EK, Gendreau M, Gendron B, Rousseau LM (eds) (2008) Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling, PATAT 2008, Université de Montréal, Montreal, CA, available online: `https://symposia.cirrelt.ca/PATAT2008/en/Proceedings`

Chalmers AF (1999) What Is This Thing Called Science. University of Queensland Press, St. Lucia, Australia

Dorigo M, Stützle T (2004) Ant Colony Optimization. MIT Press, Cambridge, MA, USA

Golden B, Raghavan S, Wasil E (eds) (2008) The Vehicle Routing Problem: Latest Advances and New Challenges, Operations Research/Computer Science Interfaces Series, vol 43. Springer

Hacking I (2001) An Introduction to Probability and Inductive Logic. Cambridge University Press, Cambridge, UK

Hooker J (1996) Testing heuristics: We have it all wrong. Journal of Heuristics 1(1):33–42

Johnson DS (2002) A theoretician's guide to the experimental analysis of algorithms. In: Goldwasser MH, Johnson DS, McGeoch CC (eds) Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol 59, American Mathematical Society, pp 215–250

Kirkpatrick S, Gelatt D, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680

Mayo DG (1983) An objective theory of statistical testing. Synthese 57:297–340

Mayo DG (1996) Error and the Growth of Experimental Knowledge. The University of Chicago Press, Chicago IL

McGeoch CC (1996) Toward an experimental method for algorithm simulation. INFORMS Journal on Computing 8(1):1–15, (this journal issue contains also commentaries by Pierre L'Ecuyer, James B. Orlin and Douglas R. Shier, and a rejoinder by C. McGeoch)

Motwani R, Raghavan P (1995) Randomized Algorithms. Cambridge University Press, Cambridge, UK

Schwefel HP (1995) Evolution and Optimum Seeking. Sixth-Generation Computer Technology, Wiley, New York NY

Van Hentenryck P, Michel L (2005) Constraint-Based Local Search. The MIT Press, Cambridge, USA

Part I
Overview

Chapter 2
The Future of Experimental Research

Thomas Bartz-Beielstein and Mike Preuss

Abstract In the experimental analysis of metaheuristic methods, two issues are still not sufficiently treated. Firstly, the performance of algorithms depends on their parametrizations—and of the parametrizations of the problem instances. However, these dependencies can be seen as means for understanding an algorithm's behavior. Secondly, the nondeterminism of evolutionary and other metaheuristic methods renders result distributions, not numbers.
Based on the experience of several tutorials on the matter, we provide a comprehensive, effective, and very efficient methodology for the design and experimental analysis of metaheuristics such as evolutionary algorithms. We rely on modern statistical techniques for tuning and understanding algorithms from an experimental perspective. Therefore, we make use of the sequential parameter optimization (SPO) method that has been successfully applied as a tuning procedure to numerous heuristics for practical and theoretical optimization problems.

2.1 Introduction

Contrary to intuition, applying and assessing randomized optimization algorithms efficiently and effectively is usually not trivial. Due to their flexibility, they fail only gradually, e.g., by using wrong parametrizations, as long as the main principle of moving towards better solutions remains intact. Statements of practitioners such as "I tried that once, it did not work" thus appear to be somewhat naïve to scientists but

Thomas Bartz-Beielstein
Institute of Computer Science, Cologne University of Applied Sciences, 51643 Gummersbach, Germany, e-mail: thomas.bartz-beielstein@fh-koeln.de

Mike Preuss
Algorithm Engineering, TU Dortmund, Germany,
e-mail: mike.preuss@tu-dortmund.de

simply reflect how the difficulties of a proper experimental evaluation are usually underestimated.

Despite these difficulties, computer scientists have all the information at hand to perform experiments. Computer experiments can be planned and randomness can be controlled, e.g., by using random seeds. This situation differs completely from the situation in field studies, e.g., in agriculture. The difficulties in the experimental analysis of randomized optimization algorithms are much more in posing the right questions than in generating empirical data. Setting up and interpreting experiments in a meaningful way is especially difficult as the methods under test are "general-purpose" optimization methods by design.

This chapter concentrates on new approaches, omitting the discussion of the "good old times" when reporting the mean best fitness value from ten runs of an algorithm was sufficient for the acceptance of an article in a journal. Looking at the history of experimentation in computer science, especially in the field of simulation and optimization, we can see that in former times simple statistics such as mean values dominated the presentations and discussions. Today, more sophisticated statistics gain more and more importance, and some journals do not accept articles which lack a sound statistical basis. State-of-the-art publications report statistically meaningful conclusions, e.g., results are based on p-values—which has been a standard in other disciplines such as medicine for many years.[1] The next step, which will be discussed in this chapter, is related to conclusions which are statistically and scientifically meaningful.

There are also some myths, e.g., statements like "Genetic algorithms are better than other algorithms (on average)", that were quoted in the 1980s, and are not heard any more. Other myths that have become extinct read: "Everything is normal", "10 is a nice number (to specify the number of runs in a repeated experiment)", and "performing good experiments is a lot easier than developing good theories."

In the meantime, approaches such as *algorithm engineering* which offer methodologies for the design, implementation, and performance analysis of computer programs, were developed (Demetrescu and Italiano 2000). Although the pen-and-paper era has been overcome in algorithm engineering (see also the discussion on p. 132 of this book), our approach differs substantially from algorithm engineering in several aspects, of which we mention only three (Bartz-Beielstein (2006) presents a comprehensive comparison):

1. The approach presented in this chapter has its origin in *design of experiments* (DOE). Ideas from Fisher (1935) have been refined and extended over recent decades. Ideas that were successfully applied in agricultural and industrial simulation and optimizations have been applied to problems from computer science. DOE has an important impact on experimentation in computer science. Most influential is Kleijnen's work. His first works go back to the 1960s. Books such as *Statistical Tools for Simulation Practitioners* (Kleijnen 1987) provide useful hints for simulation and optimization practitioners; see also his work on *Design and Analysis of Computational Experiments*, Chap. 3 of this book.

[1] A formal definition of the p-value is given in the Appendix, see p. 435.

2. The concept of *active experimentation* is fundamental to our approach. First, a very simple model is chosen—a process related to pulling oneself up by one's own bootstraps. Since the model induces a design, design points can be chosen to perform experiments. In the next step, model refinement is applied (or, if the model does not fit at all, a new model is chosen). This rather simple sequential approach is a very powerful tool for the analysis of heuristics.
3. Whereas algorithm engineering builds on the existence of theory, this is not a necessary precondition for our approach. However, obtaining general principles from experimentation may later on lead to theory. Nevertheless, this is not the primary goal. Instead, one is striving for "best-case" performance in the sense that, for a given problem and algorithm, the best possible algorithm configuration is sought.

This chapter is organized as follows: Sect. 2.2 describes specific features of experiments in computer science, especially for the analysis of computer algorithms. Two major goals are considered, namely, improving the performance of algorithms and understanding their behavior. Section 2.3 discusses three fundamental problems of experimental research: problems related (i) to the experimental setup, (ii) the interpretation of the results, and (iii) developing theories from experiments. Section 2.4 introduces the framework of the new experimentalism. Since models are central for our approach, Sect. 2.5 introduces a hierarchy of models. This section concludes with a description of *sequential parameter optimization* (SPO). [2] Pitfalls that can occur during experimentation are described in Sect. 2.6. Section 2.7 describes tools to measure and report results. This chapter concludes with a summary in Sect. 2.9.

2.2 Experimental Goals in Computer Science

Theoreticians sometimes claim that experimentation is a "nice to have" feature, but not "necessary" in computer science. As discussed in Sect. 1.2.2, there are practical problems which make experimentation necessary. There are several good reasons to further develop the methodology for experimental work:

- For many practical problems, including but not limited to black box real-valued problems, theory is far behind to nonexistent, so that there is no other means to investigate efficiency other than experiment.
- Metaheuristic optimization algorithms consist of basic working principles: Any interfaced problem can be solved "in principle." However, to attain considerable efficiency in real-world problems, it is generally needed to adapt the metaheuristic towards the problem, using any available specific knowledge about the appli-

[2] An implementation of SPO has been developed over the last years. The corresponding software package will be referred to as SPOT, an acronym which stands for *sequential parameter optimization toolbox*. SPOT is discussed in detail in Chap. 14 of this book.

cation domain. This process of adaptation is crucial but often ignored in favor of the final quality. It shall not be undertaken without a guiding methodology.

In order to justify the need for a methodology for the experimental analysis we will discuss important goals of the experimental approach.

2.2.1 Improving the Performance

Many studies use experiments to show improved algorithm performance. There are many reasons *why* we are interested in showing improved algorithm performance. One reason may be that the algorithm does not find any feasible solution. This task is related to *effectivity*. Another reason may be that we want to demonstrate that our heuristic is competitive to the best known algorithm. This task is related to *efficiency*. If our focus of the experimental analysis lies in efficiency, then we are considering *tuning* problems.

Effectivity and efficiency require different experimental setups. To analyze *effectivity*, we consider several problem instances, e.g., different starting points, dimensions, or objective functions. To study *efficiency*, we are seeking an improved parameter set for the algorithm, while keeping the problem instance constant. This leads to a generic classification of the experimental parameters: the first class comprises parameters related to the algorithm, whereas the second class consists of parameters related to the problem.

a) *Algorithm parameters* are related to the algorithm, which should be improved. A set of algorithm parameters will be considered as an *algorithm design*. Note that an algorithm design is a set, which can be empty or contain infinitely many algorithm parameters.
b) *Problem parameters* describe the problem (instance) to be solved by the algorithm. A *problem design* consists of none, one or several problem parameters.

Example 2.1. An algorithm design includes parameters such as population size or selection strength. A problem design comprises search space dimensions, starting points, or objective functions. □

Regarding the problem design and its associated *budget*, at least two different situations can be distinguished:

a) *Real–world settings:* In these situations, complex objective functions, e.g., from simulation, occur. Generally, only a small number of function evaluations is possible.
b) *Theoretical settings:* In these situations, simple objective functions are used. These, partially artificial, functions should provide some insight into the algorithms's behavior. Generally, many function evaluations are possible.

2.2.2 Understanding

Until now, we have investigated ways to improve algorithm performance, i.e., improving efficiency (tuning) or effectivity (robustness). In several situations, we are interested in *why* an algorithm performs poorly or well. Methods for understanding its elements, e.g., procedures, which can be modified by parameters, are presented next.

Screening, e.g., in order to detect the most influential parameters or *factors*, is useful in real-world and in theoretical situations. Statistically speaking, we are interested in the effects of the parameters on the outcome. This is related to important questions: "Does a parameter influence the algorithm's performance?" or "How to measure these effects?" Consider a first model: $Y = f(X)$, where $X = (X_1, X_2, \ldots, X_r)$ denotes r parameters from the algorithm design, and Y denotes some output (i.e., the best function value from 1000 evaluations). If the problem design remains unchanged, we can perform:

1. *Uncertainty analysis*, i.e., we compute the average output, standard deviation, and outliers. Therefore, uncertainty analysis is related to the outcome Y.
2. *Sensitivity analysis*, i.e, we analyze which of the factors are most important in influencing the variance in the model output Y. Therefore, sensitivity analysis is related to the input values, i.e., the relationship between X_i, X_j, and Y.

To measure parameter effects, mathematics and statistics provide different approaches, namely derivation, analysis of variance, and regression. Although these approaches look different at first sight, they have very much in common as will be seen later on. SPOT's screening phase comprises short runtimes, sparse designs, extreme values, and outliers that destroy the SPOT metamodel; see also the discussion and examples in Chap. 14.

There is no general answer to the question "How many factors are important?" However, the following rule is observed by many practitioners: The input factor importance is distributed as the wealth in nations—a few factors produce nearly all the variance.

2.3 Problems

Now that we have discussed relevant tasks for the experimental analysis of computer algorithms, we consider severe problems related to experimentation. Obviously, performing an experimental study requires more than reporting results from a few algorithm runs. Generating scientifically meaningful results is a very complex task. We will present three fundamental problems that give experimenters a hard time, namely problems related to:

1. The experimental setup
2. Interpreting the significance of the results

3. High-level theory

2.3.1 Problems Related to the Experimental Setup

An analysis of recent publications makes evident that there is still space for improvement of up-to-date approaches in experimentation. Example 2.2 illustrates our considerations.

Example 2.2. Can the following be considered good practice? The authors of a journal article used 200,000 function evaluations as the termination criterion and performed 50 runs for each algorithm. The test suite contained 10 objective functions. For the comparison of two algorithms, population sizes were set to 20 and 200. They used a crossover rate of 0.1 in algorithm A, and 1.0 in B. The final conclusion from their experimental study reads: "Algorithm A outperforms B significantly on test problem f_6 to f_{10}." □

Problems related to this experimental study may not be obvious at first sight, but consider the following:

1. Why did the authors use 200,000 function evaluations to generate their results? Would results differ if only 100,000 function evaluations were used?
2. Why did they use 50 runs for each algorithm? Again, would 10 or 100 repeats result in different conclusions?

Solutions for the first question are discussed in some publications. Cohen (1995) presents a comprehensive discussion of so-called floor and ceiling effects, and Hoos and Stützle (2005) develop statistical tools, so called *run-length distributions*. Problems such as mentioned in the second question, the number of repeats, will be detailed in Sect. 2.3.2. They are related to p-values, or more abstractly, significance (Morrison and Henkel 1970). Example 2.2 demonstrates that we need tools to determine:

1. Adequate number of function evaluations, especially to avoid floor or ceiling effects
2. Thoughtfully chosen number of repeats
3. Suitable parameter settings for comparison
4. Suitable parameter settings to get working algorithms

2.3.2 Problems Related to the Significance of Experimental Results

In the following, we will focus on comparisons of results from experimental studies. High-quality tools from statistics are used in recent publications. Nowadays, it is no problem to generate experimental data and to produce statistics. However, there are nearly no tools to interpret the scientific significance of these statistical results. The reader may claim that more and more authors use statistical tests to validate the significance of their results. However, results based on tests can be misleading.

We will consider hypotheses testing first. A hypothesis is a statement made about a population, e.g., with respect to its mean. Acceptance or rejection of the hypothesis is based on a test statistic T on a sample taken X from the population. Hypothesis testing is a rule of inductive behavior: Accept/reject are identified with deciding to take specific actions (Mayo and Spanos 2006a). The process of hypothesis testing consists on taking a random sample from the population and making a statistical hypothesis about the population. If the observations do not support the claim postulated, the hypothesis is rejected. Associated with the decision is the significance level α. Assuming normal distributed random variables, the procedure for hypothesis testing involves the following five steps:

1. State a null hypothesis, H_0. For example, H_0: $\mu_A - \mu_B = 0$, i.e., we hypothesize that the mean function value μ_A of algorithm A and the mean function value μ_B of algorithm B are the same. If H_0 is true, any observed difference in means is attributed to errors in random sampling.
2. Declare an alternate hypothesis, H_1. For the example under consideration, it could be H_1: $\mu_A - \mu_B > 0$, i.e., we are considering a one-sided test. Since smaller function values are preferred (minimization), this indicates that algorithms B performs better than A.
3. Specify a test statistic, T. For claims about a population mean from a population with a normal distribution, if the standard deviation σ is known, the appropriate significance test is known as the z-test. The test statistic is defined as
$$z_0 = \frac{\overline{x} - \mu_0}{\sigma_x}, \tag{2.1}$$
where $\sigma_x = \sigma/\sqrt{n}$ is the standard error. In the example under consideration, T will be based on the difference of observed means, $\overline{X}_A - \overline{X}_B$. Using (2.1), the test statistic follows the standard normal distribution (with mean = 0 and standard deviation = 1).
4. Define a rejection region (the critical region, R) for T based on a pre-assigned significance level α.
5. Use observed data, e.g., $\overline{x}$, to determine whether the computed value of the test statistic lies within or outside the critical region. The quantity we are testing is statistically significant at the $\alpha\%$ level, if the the test statistic is within the critical region, see also (A.1) on p. 433. The significance level α indicates how much we are willing to risk (in terms of probability) an incorrect rejection of the null hypothesis when the latter is true. Figure 2.1 illustrates this procedure.

In hypothesis testing we use the terms errors of Type I (α) and Type II (β) to define the cases in which a true hypothesis is rejected or a false hypothesis is accepted (not rejected), respectively. The complement of β is called the power of the test of the null hypothesis H_0 versus the alternative H_1. Using a sample of size n with mean $\overline{x}$ and standard deviation σ, the z-statistic, cf. (2.1), can be calculated. The p-value for a one-sided test is defined as: p-value$= \Pr(Z > z_0)$. The p-value

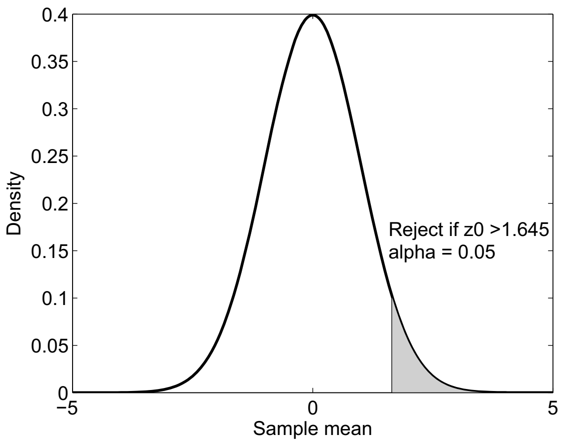


Fig. 2.1: Hypothesis testing. Consider $n = 100$, $\sigma = 1$, $\mu_0 = 0$, and $\alpha = 0.05$. The critical region is $[1.645, \infty[$, and $z_0 = \overline{x} \times 10$, cf. (2.1). For instance, the testing rule leads to a rejection of the null hypothesis if the difference of observed means is larger than 0.1645

associated with z_0 is compared to α to decide whether or not to reject the null hypothesis. The criteria to use for hypothesis testing is: Reject H_0 if the p-value is smaller than α, otherwise do not reject H_0.

Next, we consider two typical problems related to the p-value.

2.3.2.1 Misinterpretation

First, we consider a typical problem related to the definition of the p-value. Since the null hypothesis is either false or true, the p-value is not the probability that H_0 is true. The p-value is defined as[3]

$$p = \Pr\{\text{result from test-statistic, or greater} \mid \text{null model is true}\}. \tag{2.2}$$

A typical misinterpretation of the p-value reads as follows:

$$p = \Pr\{\text{ null model is true} \mid \text{test-statistic }\}.$$

However, the p-value has “no information to impart about the verity of the null model itself” (Gregoire 2001). To illustrate this difference, the reader may compare $\Pr\{A|B\}$ and $\Pr\{B|A\}$, where A denotes the event “born on 1st January, 1980” and B represents “gender female.”

[3] See also the description on p. 435 in the Appendix of this book.

2.3.2.2 The Large n Problem

Second, we will exemplify one typical problem that has direct consequences for the experimental analysis of optimization algorithms. It is fundamental to all comparisons—even to high-level procedures. To perform experiments in computer science, the following steps are to be done:

1. Select test problem (instance).
2. Run algorithm A, say n times, which results in n function values.
3. Run algorithm B, say n times, which gives n function values.
4. Calculate the difference of the observed means, say $\overline{X} = \overline{X}_A - \overline{X}_B$.

This sounds trivial, but there are many implications which arise from the specification of n, the number of repeats. Recall, that n was set to 50 in Example 2.2.

Example 2.3 (Hypotheses testing). We perform a comparison of two algorithms A and B. The null hypothesis $H_0 : \mu = 0 \ (= \mu_0)$, is tested against the alternative hypothesis, $H_1 : \mu > 0$ at a level of confidence of 95%, i.e., $\alpha = 0.05$, using a sample of size $n = 5$ with a difference of the observed means $\overline{x} = 0.1$ and a (known) standard deviation $\sigma = 1.0$. Based on this setup, the experimenter is interested in demonstrating that algorithm B outperforms A. Performing $n = 5$ runs, the p-value reads $p = 0.4115$, which is not very convincing. Maybe additional experiments are helpful? Increasing the number of repeats from $n = 5$ to 10 gives $p = 0.3759$, which is better, but not convincing. However, the direction for further experimentation is clear: $n = 100$ gives $p = 0.1587$, and $n = 1000$ produces the desired result: $p = 0.0008$. The experimenter concludes his article with the words: "My newly developed algorithm B clearly outperforms the known-best algorithm A."

This situation is illustrated in Fig. 2.2, where the p-value is plotted versus the number of runs. This result is not surprising, since (2.1) depends on n, i.e., the number of samples. □

We may ask the reader if the result from Example 2.3, i.e., "B clearly outperforms A" is:

a) Statistically correct
b) Scientifically meaningful

After introducing the new experimentalism in Sect. 2.4, this problem will be reconsidered.

2.3.3 *Problems Related to High-Level Theory*

In computer science, the Popperian view based on falsification is an, if not *the*, established definition of science (Popper 1959). Once proposed, speculative claims (or theories) are to be rigorously tested by experiment and observation. If they fail these

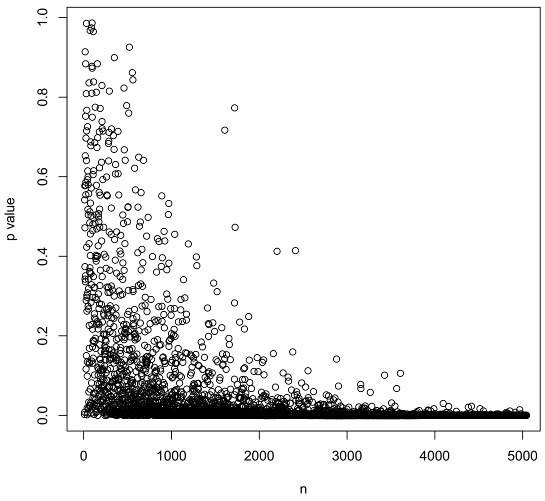


Fig. 2.2: Increasing the number of experiments can reduce the p-value. This figure illustrates the situation described in Example 2.3. The number of repeats is varied from $n = 10$ to 5000. For each setting, n realizations of the random variables Y_A and Y_B are generated and the corresponding p-value is determined

tests, theories must be eliminated and can be replaced by new theories or speculative conjectures. However, the philosophy of science, i.e., the discipline in which Popper's ideas were established, has made some progress over the last decades. New approaches, which strengthen the role of experiments, have gained importance.

Since these discussions have relevant impact on the meaning of experimentation and its relation to theory in computer science, we will sketch out significant aspects of this debate. Chalmers (1999) notes, that it "is ironic, ... , that Popper, who at times characterized his own approach with the slogan 'we learn from our mistakes', failed precisely because his negative, falsificationist account did not capture an adequate, positive account of how science learns from mistakes (falsifications)." Popper states that theory formulates hypotheses which we accept as knowledge insofar as they can withstand severe experiments set up to falsify them (Popper 1959). However, an important dilemma that arises from the Popperian view can be formulated as follows:

> It is impossible to decide whether an hypothesis itself or its supporting assumption is wrong. Moreover, meaningful assumptions require additional assumptions, which leads to an infinite regress.

Since we cannot discuss all problems related to the Popperian view, the interested reader "who wants to obtain a firmer grasp of one of the most important yet simultaneously least understood developments of all time" (Champion 2009) is referred to Chalmers (1999). However, we can present alternatives and recent approaches. Therefore, we will clarify the role of experiments in computer science and present relevant tasks which can be treated by experiments.

2.4 The New Experimentalism

In the previous section, problems related to the experimental setup, significance of the results, and high-level theory were presented. A debate devoted to these central questions has not yet started in computer science. The reader may answer the question: Obviously, astronomy is considered scientific, and astrology not—but, what about experimental research? Fortunately, this debate is going on in other scientific disciplines, e.g., in the philosophy of science. We claim, that in computer science a similar debate is necessary to provide a scientific foundation for experimental research. Therefore, we will introduce the *new experimentalism*, which is the most influential trend in recent philosophy of science (Ackermann 1989). The new experimentalists are seeking a relatively secure basis for science, not in theory or passive observation, but in active experimentation.

When free from error, experimental results may confirm scientific claims. However, more than validations of scientific claims can be obtained from experiments. An experiment which serves to detect an error in some previously stated assertion serves as a positive as well as a negative function. Experiments do not only serve as falsifications of certain assertions, they also positively identify previously unknown effects.

The key idea in this concept, which can be seen as an extension of the Popperian view, is that a claim can only be said to be supported by experiment if the various ways in which the claim could be false have been analyzed and eliminated. The new experimentalists will ask whether the confirmations would have been likely to occur if the theory were *false*. Mayo (1996) describes a detailed account of how experimental results can be reliably established using error statistics and error-eliminating techniques. Many of the ideas presented in this book describe how statistics can be used to obtain valid experimental results. Considering ideas from the current discussion in the philosophy of science may be inspiring for further research in computer science and may be helpful to bridge the gap between theory and experiment, hopefully leading to—as stated by Chalmers (1999)—a "happy meeting of theory and experiment." Since models are the key elements of a scientific framework for experimental research, they will be discussed in the following section.

2.5 Experimental Models

2.5.1 The Role of Models in Science

The standard interpretation of models is derived from mathematical logic. Scientific reasoning is to a large extent model-based reasoning. Forty years ago, Suppes (1969) published his much cited paper "A Comparison of the Meaning and Uses of Models in Mathematics and the Empirical Science." He claims that the meaning and the use of models can be interpreted as being the same in the empirical sciences as it is in mathematics, and, more particularly, in mathematical logic. This view can be considered as the standard view within science. Giere (1999) claims that this standard interpretational (or instantial) view of models is not adequate for empirical science. He proposes a *representational* view of models which is more suited for the needs of empirical investigations. To understand why instantial models are not adequate, we consider Suppes' definition of models, which is based on the definition of theories: "A theory is a linguistic entity consisting of a set of sentences and models are non-linguistic entities in which the theory is satisfied." A model $\mathcal{A}$ is a set-theoretical structure consisting of a set of objects together with properties, relations, and functions defined over $\mathcal{A}$. A model enables the interpretation of a set of uninterpreted axioms.

Example 2.4. As an example we consider models used for non-Euclidean geometry, especially for hyperbolic geometry. Non-Euclidean geometry systems differ from Euclidean geometry in that they modify the parallel postulate (Euclid's fifth postulate). In general, there are two forms of (homogeneous) non-Euclidean geometry: Hyperbolic geometry and elliptic geometry.

There are four models commonly used for hyperbolic geometry: the Klein model, the Poincaré disc model, the Poincaré half-plane model, and the Lorentz model. These models define a real hyperbolic space which satisfies the axioms of a hyperbolic geometry. We will illustrate two of these models in the following, see Fig. 2.3.

1. The Klein model uses the interior of a circle for the hyperbolic plane. Hyperbolic lines are chords of this circle.
2. The hyperbolic plane in the Poincaré half-plane model is defined as one-half of the Euclidean plane, as determined by a Euclidean line h. The line h itself is not included. Either rays perpendicular to h or half-circles orthogonal to h are hyperbolic lines.

□

As in this example, logicians consider mainly models that consist of abstract entities, e.g., lines or circles. In principle, these objects could be physical objects. As shown in Example 2.4, uninterpreted logic formula may be interpreted using many different instantial models which are isomorphic. This isomorphism is called an analogy. So, we can consider many analogue models for one theory, see Fig. 2.4.

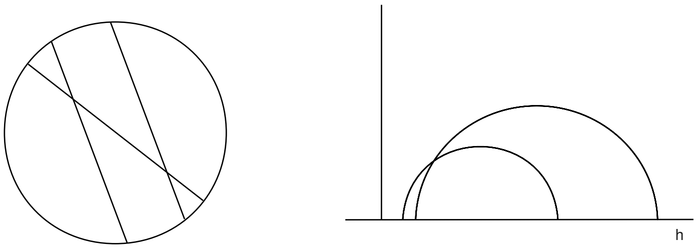


Fig. 2.3: Two models for hyperbolic geometry. *Left:* The Klein model. *Right:* The Poincaré half-plane model. Hyperbolic lines are chords of the circle in the Klein model; they are rays or half-circles in the Poincaré half-plane model

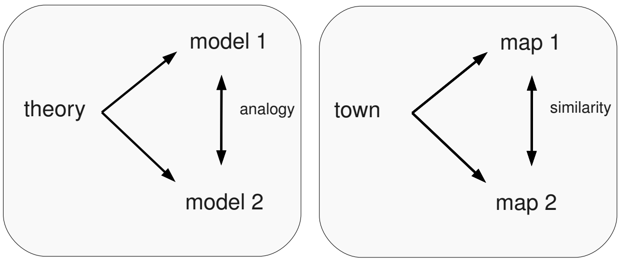


Fig. 2.4: *Left:* Analogy interpreted as isomorphism. *Right:* Similarity as a concept for real-world (representational) models. Maps, which represent the same geographical area, are similar, not necessarily isomorphic. Consider, e.g., a map which illustrates railway lines and a map which illustrates the climate

2.5.2 Representational Models

Computer programs consist of several hundred or thousand lines of code. Several programs use special features of the operating system, which depends on the underlying hardware. Hence, modeling computer programs requires more complex models than models from logic. Following Giere (1999), we consider *representational models* for real-world scenarios. Maps are good examples of representational models. One object (map) represents another real-world object (geographical region). In contrast to models in theory, representational objects are *similar*, not isomorphic. These similarities are context dependent. At first sight, similarity seems to be a much weaker property than analogy. This is only relative, because in any particular context, we can specify what is said to be similar to what, in what ways, and to what extent.

Next, we will consider mathematical models which describe properties of computer algorithms. For example, the equation

$$T(n) = \frac{1}{2}(n^2 + n) \tag{2.3}$$

is a linguistic object which describes the (worst-case) time complexity function T of an algorithm, e.g., quick sort, as a function of the problem dimension n. Nobody would claim that Eq. 2.3 is the true running time (wall time) of quicksort as nobody would claim that the equation $y(t) = y_0 + vt$ describes the exact position of a real object, e.g., a car, because no real object can maintain a constant velocity in a perfect straight line. One possible way to resolve this problem is to introduce error into Eq. 2.3. Although technically correct, it is not common practice and in many situations not necessary to use error terms. Error equations are useful to compare the abstract model with reality. This comparison can be performed at the beginning and at the end of an experimental study. Then, the error terms can be used to determine the degree of similarity between the abstract model and the real algorithm.

Mathematical models can provide information about the degree of similarity with real systems by interpreting error equations as hypotheses about particular algorithms or systems in the real world. These hypotheses may be judged true or false based on active experimentation. Statistical testing plays a central role in establishing this scientific approach to experimentation.

After considering suitable models for the experimental analysis of algorithms, we have to bridge the gap between the data obtained by experimentation and the theory. Obviously, experimental data can constitute only a finite sample of outputs from the experimental system, whereas scientific hypotheses consider an infinite number of cases. This leads to the problem of how to link the problem-specific experimental data to the primary theoretical hypotheses. Introducing a hierarchy of models can provide a solution to this problem. Testing the fit of a model to the real object or system, we do not compare the model with data but with a model of data. Experiments are used to judge the similarity between the higher level and the real system. These experiments require the specification of models, too. We describe this framework of models in the following.

2.5.3 A Framework of Models

We obtain a series of conceptual representations or models ranging from the primary scientific hypotheses to the details of generating data. Mayo (1996, p.129) introduces:

1. Models of primary scientific hypotheses
2. Models of experiment

Procedure 2.1: (1+1)-ES()

```
$t := 0$;
initialize($\mathbf{x}$, $\sigma$);
$y_p := f(\mathbf{x_p})$;
repeat
    $\mathbf{x_o} := \mathbf{x_p} + \sigma(\mathcal{N}(0,1), \mathcal{N}(0,1), \ldots, \mathcal{N}(0,1))^T$;
    $y_o := f(\mathbf{x_o})$;
    if $y_o \leq y_p$ then
        $\mathbf{x_p} := \mathbf{x_o}$;
        $y_p = y_o$;
    end
    modify $\sigma$ according to $1/5$th rule;
    $t := t + 1$;
until TerminationCriterion() ;
return $(\mathbf{x_p}, y_p)$
```

3. Models of data

Primary models are means to break down a substantive inquiry into one or more local questions that can be probed reliably. *Experimental models* are used to relate primary questions to canonical questions about the particular type of experiment at hand. They can be used to relate the data to the experimental questions. Finally, *data models* are applied to generate and model raw data so as to put them in canonical form and to check if the actual data generation satisfies various assumptions of the experimental models. An adequate account of experimental testing must not begin at the point where data and hypotheses are given, but must explicitly incorporate the intermediate theories of data, instruments, and experiment that are necessary to obtain experimental evidence in the first place. We will present an example to illustrate these different models.

Example 2.5. We consider a simple *evolution strategy* (ES), the so-called (1+1)-ES, see Procedure 2.1. The $1/5$th rule states that σ should be modified according to the rule

$$\sigma(t+1) := \begin{cases} \sigma(t)a, & \text{if } P_s > 1/5 \\ \sigma(t)/a, & \text{if } P_s < 1/5 \\ \sigma(t), & \text{if } P_s = 1/5 \end{cases} \tag{2.4}$$

where the factor a is usually between 1.1 and 1.5 and P_s denotes the success rate (Beyer 2001). The factor a depends particularly on the measurement period g, which is used to estimate the success rate P_s. During the measurement period, g remains constant. For $g = n$, where n denotes the problem dimension, Schwefel (1995) calculated $1/a \approx 0.817$. Beyer (2001) states that the "choice of a is relatively uncritical" and that the $1/5$th rule has a "remarkable validity domain." He also mentions limits of this rule.

Based on these theoretical results, we can derive certain scientific hypotheses. One might be formulated as follows: *Given a spherical fitness landscape, the (1+1)-ES performs optimally, if the step-sizes σ is modified according to the $1/5$th rule as stated in Eq. 2.4.* This statement is related to the primary model.

In the experimental model, we relate primary questions or statements to questions about a particular type of experiment. At this level, we define an objective function, a starting point, a quality measure, and parameters used by the algorithm. These parameters are summarized in Table 2.1.

Table 2.1: $(1+1)$-ES parameters. The first three parameters belong to the algorithm design, whereas the remaining parameters are from the problem design (see Sect. 2.2.1)

Name	Symbol	Factor name in the algorithm design
Initial stepsize	$\sigma(0)$	S0
Stepsize multiplier	a	A
History	$g = n$	G
Starting point	$\mathbf{x_p}$	
Problem dimension	n	
Objective function	$f(\mathbf{x}) = \sum x_i^2$	
Quality measure	Expected performance, e.g., $E(y)$	
Initial seed	s	
Budget	$t_{\max}$	

Data from these experiments are related to an experimental question, which can be stated as follows: *Determine a value for the factor a, such that the (1+1)-ES performs best with respect to the quality measure specified in Table 2.1. And, is this result independent of the other parameters presented in Table 2.1?* Note that Table 2.1 contains all factors that are used in this experiment.

Finally, we consider the data model. A common practice in statistics is to seek data models that use similar features and quantities of the primary hypothesis. For example, if the primary model includes questions related to the mean value μ of some random variable X, then the data model might use the sample mean $\overline{x}$. Here, we are interested in the expected performance of the (1+1)-ES. Thus, we can use the average from, say fifty, repeats, $\overline{y_i}$, to estimate the expected performance. In addition, standard errors or confidence intervals can be reported. So, in the data model, raw data are put into a canonical form to apply analytical methods and to perform hypothesis tests. How one scientific claim, which is related to the primary model, can be broken down into local claims, is illustrated in Fig. 2.5. □

Based on these models, we can apply statistics to draw conclusions and learn from experimental results. The refined relationship between theory and model is shown in Fig. 2.6.

2.5.4 Mayo's Learning Model

The classical approach in statistical inference can be interpreted as a means of deciding how to behave. To contrast her formulation testing with this behavioristic

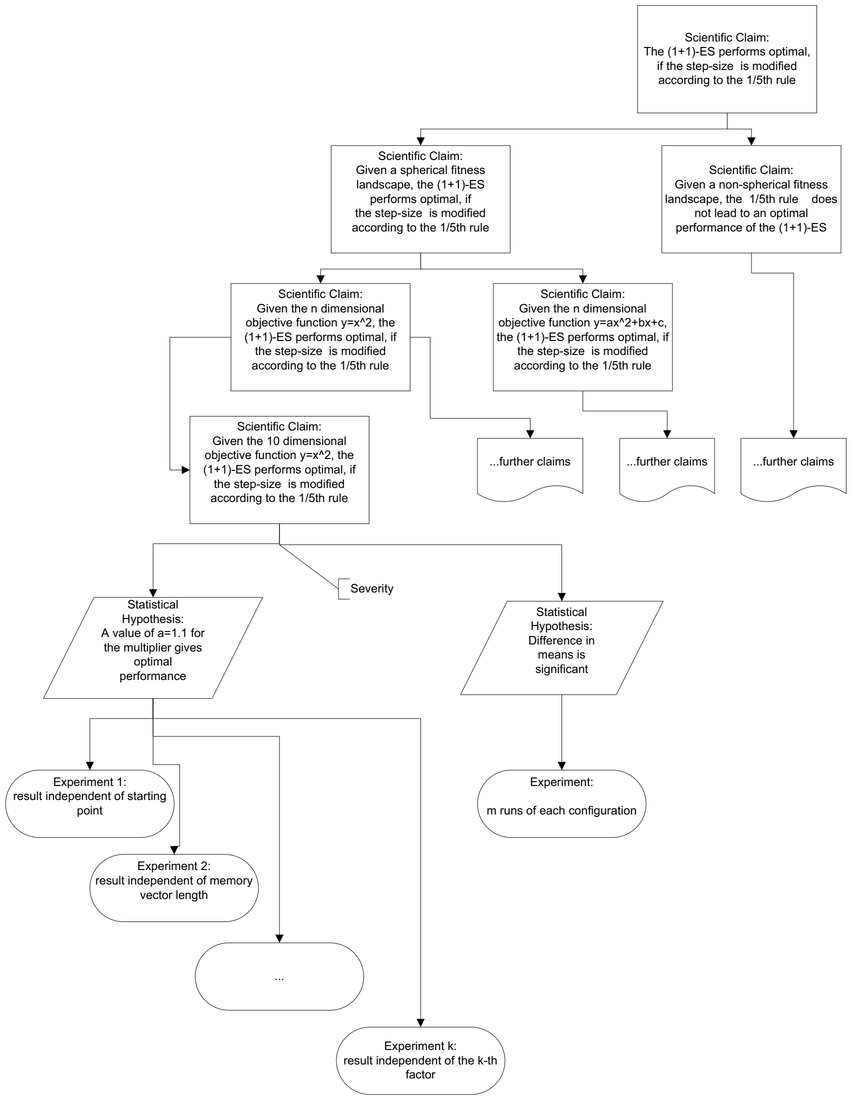

Scientific Claim:
The (1+1)-ES performs optimal, if the step-size is modified according to the 1/5th rule
Scientific Claim:
Given a spherical fitness landscape, the (1+1)-ES performs optimal, if the step-size is modified according to the 1/5th rule
Scientific Claim:
Given a non-spherical fitness landscape, the 1/5th rule does not lead to an optimal performance of the (1+1)-ES
Scientific Claim:
Given the n dimensional objective function y=x^2, the (1+1)-ES performs optimal, if the step-size is modified according to the 1/5th rule
Scientific Claim:
Given the n dimensional objective function y=ax^2+bx+c, the (1+1)-ES performs optimal, if the step-size is modified according to the 1/5th rule
Scientific Claim:
Given the 10 dimensional objective function y=x^2, the (1+1)-ES performs optimal, if the step-size is modified according to the 1/5th rule
...further claims
...further claims
...further claims
Severity
Statistical Hypothesis:
A value of a=1.1 for the multiplier gives optimal performance
Statistical Hypothesis:
Difference in means is significant
Experiment 1:
result independent of starting point
Experiment 2:
result independent of memory vector length
...
Experiment k:
result independent of the k-th factor
Experiment:
m runs of each configuration

Fig. 2.5: Primary scientific hypothesis broken into local hypotheses. Rectangles denote elements from the primary model, elements from the experimental model use parallelograms, and ellipses are used to represent elements from the data model

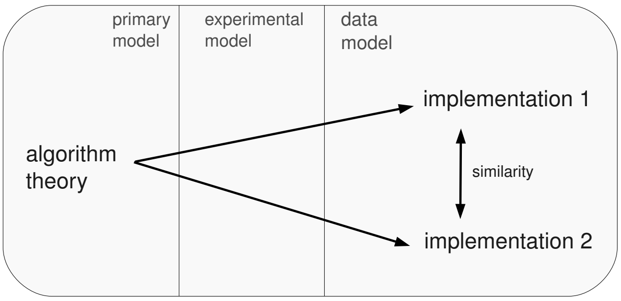


Fig. 2.6: Refining the theory–model relation from Fig. 2.4 by introducing a model hierarchy

model, Mayo (1983) introduces the term *learning model*. The distribution of the test statistic is used to control error probabilities. Statistical tests are seen as "means of learning about variable phenomena on the basis of limited empirical data."

Consider a statistical model with some unknown parameter θ. Mayo (1983) introduces tools for specifying tests that "will very infrequently classify an observed difference as significant (and hence reject H) when no discrepancy of scientific importance is detected, and very infrequently fail to do so (and so accept H) when θ is importantly very discrepant from θ_0." A discussion of the *severity* interpretation of acceptance and rejection can be found in Mayo and Spanos (2006b). They demonstrate how error-statistical tools can extend commonly used pre-data error probabilities (significance level and power) by using post-data interpretations (severity) of the resulting rejection or acceptance. Regarding the (1+1)-ES described in Example 2.5, we have to ensure that the recommended value for the factor a is independent of the initial stepsize, problem dimension, seed, etc.

Figure 2.7 gives an overview of effects which should be considered in the experimental approach. It represents factors which are a) expected to have an effect, b) should have no effect with a high probability, and c) which should have no effect at all.

2.5.5 Sequential Parameter Optimization

2.5.5.1 Definition

Now that we have introduced primary, experimental, and data models, we are ready to introduce the SPO framework, which is built on these elements.

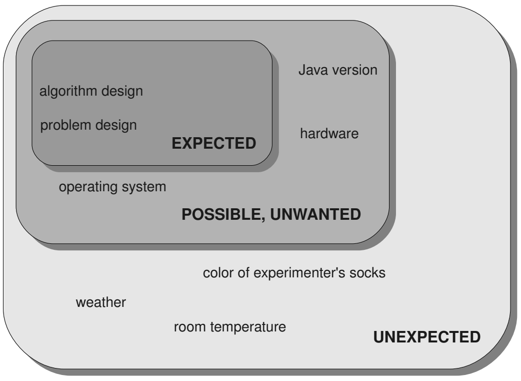


Fig. 2.7: Classification of factors in the experimental approach. a) Factors which belong to the algorithm and problem design should have an effect on the experimental outcome. b) Sometimes, side-effects or unwanted effects occur during experimentation. The experimenter should eliminate the influence of these factors. If this is not possible, these factors should be included in the set of factors used for active experimentation (algorithm or problem design). The experimenter can determine their (error) probabilities. c) Finally, there are infinitely many factors that "obviously" have no effect on the result, e.g., the color of the experimenter's socks. Since these are infinitely many factors, it is impossible to test all of them. However, sometimes it may be interesting to include one factor into the experimental design—or, as the physicist George Darwin used to say, "every once in a while one should do a completely crazy experiment, like blowing the trumpet to the tulips every morning for a month. Probably nothing would happen, but what if it did?"

Definition 2.1 (Sequential Parameter Optimization). *Sequential parameter optimization* (SPO) is a framework for tuning and understanding of algorithms by active experimentation. SPO employs methods from error statistics to obtain reliable results. It comprises the following elements:

SPO-1:	Scientific questions	SPO-3:	Experiments
SPO-2:	Statistical hypotheses	SPO-4:	Scientific meaning

□

These elements can be explained as follows.

2.5.5.2 Scientific Questions

Starting point of the investigation is a scientific question which is related to the primary model. This question often deals with assumptions about algorithms, e.g., influence of parameter values or new operators.

2.5.5.3 Statistical Hypotheses

This (complex) question is broken down into (simple) statistical hypotheses for testing. Mayo (1996, p. 190) writes:

> Our approach to experimental learning recommends proceeding in the way one ordinarily proceeds with a complex problem: break it into smaller pieces, some of which, at least, can be tackled. One is led to break things down if one wants to learn [...] Setting out all possible answers to this one question becomes manageable, and that is all that has to be "caught" by our not-H.

2.5.5.4 Experiments

Using the experimental model, the following steps are performed. For each hypothesis:

a) Select a model (e.g., regression trees) to describe a functional relationship.
b) Select an experimental design.
c) Generate data. This step is related to the data model.
d) Refine the model until the hypothesis can be accepted/rejected.

In Chap. 14 we will present an implementation of steps a) to d), which can be performed sequentially. The corresponding software programs will be referred to as SPOT.

2.5.5.5 Scientific Meaning

To assess the scientific meaning of the results from this experiment conclusions are drawn from the hypotheses. Here, the concept of severity suggested in Mayo (1996) comes into play: "Stated simply, *a passing result is a severe test of hypothesis H just to the extent that it is very improbable for such a passing result to occur, were H false.*" Severity provides a meta-statistical principle for evaluating proposed statistical inferences. It tells us how 'well probed' (not 'how probable') hypotheses are and is an attribute of the test procedure as a whole. Once the data x_0 of the test T are available, they enable us to evaluate the severity of the test and the resulting inference.

Consider $\mu_1 = \mu_0 + \delta$, where δ is the discrepancy that is warranted. The experimental result is denoted as $\overline{x}$. Mayo and Spanos (2006a) present an explicit formula for determining the severity with which test T passes $\mu_1 > \mu$ in case of a statistically significant result, i.e., reject H_0 with data x_0. The severity can be determined as

$$\begin{aligned} SEV(\mu > \mu_1) &= \Pr(T(X) \leq T(x_0); \mu > \mu_1 \text{ false}) \\ &= \Pr(T(X) \leq T(x_0); \mu \leq \mu_1). \end{aligned}$$

It follows:

$$SEV(\mu > \mu_1) = \Pr\left(Z \leq \frac{\overline{x} - \mu_1}{\sigma_x}\right) \tag{2.5}$$
$$= 1 - \Pr\left(Z > \frac{\overline{x} - \mu_1}{\sigma_x}\right)$$

with $Z \sim \mathcal{N}(0, 1)$.

2.5.5.6 Example

Typical SPO steps are shown in Example 2.6.

Example 2.6. SPO-1: Scientific question: The (1+1)-ES performs optimal, if the step-size is modified according to the 1/5th rule. This scientific claim can be divided into subclaims, e.g., the general term "spherical functions" is replaced by one instance, say "sphere function." This is done until a clearly specified experiment is defined and statistical hypotheses can be formulated.

SPO-2: Statistical hypotheses can be formulated as follows:

H_1: Consider the (1+1)-ES on the 10-dimensional sphere function with parameters from Table 2.1. A value of $a = 1.1$ for the multiplier gives optimal performance. Note that each hypothesis requires the specification of the related algorithm and problem designs to enable reproducibility. The full specification includes also the selection of an appropriate performance measure. We have chosen the average function value from $n = 50$ repeats, but other measures are possible. The corresponding null hypothesis (H_0) reads: There is no such effect (optimal performance) for $a = 1.1$.

SPO-3: For hypothesis H_1:

a) A regression model is chosen.
b) Since experimentation is not expensive, a space-filling design with two repeats of each design point is selected as the initial design in this case.
c) Algorithm runs are performed with parameter settings from the initial design.
d) Sequential runs are performed until the model allows statements about the effects of parameters (of the algorithm design in Table 2.1) on the function value predictions.

SPO-4: Similar results from different hypotheses, etc. indicate the scientific relevance of the results. The reader is also referred to Mayo and Spanos (2006b) and Bartz-Beielstein (2008).

□

2.5.6 The Large n Problem Revisited

Based on (2.5), the large n problem, which was introduced in Sect. 2.3.2.2, can be reconsidered. In the following, a difference in means of $\delta = 0.25$ is considered scientifically meaningful. The same setting as in Example 2.3 is analyzed.

Example 2.7 (Hypotheses testing and severity). We test the null hypothesis H_0 : $\mu = 0\ (= \mu_0)$, against the alternative hypothesis, $H_1 : \mu > 0$ at a level of confidence of 95%, i.e., $\alpha = 0.05$, using a sample of size $n = 5$ with a mean $\overline{x} = 0.1$ and a (known) standard deviation $\sigma = 1.0$. Based on this setup, the experimenter is interested in demonstrating that B outperforms A. Performing $n = 1000$ runs, the p-value reads $p = 0.0008$. How severely does test T pass $\mu > 0.25$ with this result? The answer is $SEV(\mu \geq \mu_1) = 0.0$. □

Note, $SEV(\mu \geq \mu_1) = 0.0668$ for $n = 100$, i.e., severity increases for smaller values of n. An α-significant difference with large n passes $\mu_1 > \mu$ less severely than with a small n. Mayo and Spanos (2010) conclude: " With this simple interpretive tool, all of the variations on 'large n criticisms' are immediately scotched."

We have presented the SPO as one possible framework for an experimental analysis of computer algorithms. Next, we will discuss problems that are not restricted to SPO, but also relevant to other experimental approaches for the analysis of algorithms. The issues treated are related to data generation, namely tasks and setups of experiments, and to organizational aspects of meaningful experimentation.

2.6 Pitfalls of Experimentation with Randomized Algorithms

Experimentation in algorithm engineering deals mostly with deterministic methods. It seems reasonable to argue that this type of experiment has fewer "degrees of freedom" than is usual in randomized optimization algorithms, namely metaheuristics. However, one can still make many different types of mistakes, leading to explicit guidelines for experimentation such as the one provided by Johnson (2002).

When using randomized optimization techniques as evolutionary algorithms, several more problems emerge which do not exist for deterministic methods. First of all, the metaheuristic perspective is somewhat reversed compared with the perspective used in algorithm engineering:

- In algorithm engineering, the focus is on solving a specific problem or problem class. For this purpose, existing algorithms are modified or new ones designed. The performance is experimentally compared with the predictions made by theory. Where necessary, the algorithms are modified, the theory adapted, and the algorithm engineering cycle is started anew. In any case, one is usually most interested in provable algorithm properties, namely convergence speed. The established algorithms are not designed to perform well on different problems.

- In metaheuristics, one designs algorithms for very general (often black-box) problems and then investigates for which problem classes the algorithms perform well. Modifications may be in order for any specific application, but the main algorithmic frame remains unchanged. One may also start with a concrete problem to solve, take a metaheuristic "off the shelf," and improve its performance by means of intuition and iterated experimentation, using available or experimentally obtained problem knowledge. Theory is not available in many cases, as the problems treated and algorithms employed are often too complicated and the stochastic algorithm behavior makes it very difficult to grasp it theoretically. Where not possible, one may however try to develop theory on a reasonable simplification of the original algorithm—this could be termed "algorithm reengineering" (Jägersküpper and Preuss 2008). The main focus however is on the ability of the algorithms to approximate concrete real-world or benchmark problems well enough in reasonable time, not on provable algorithm properties.

In the following, we mainly deal with the specific pitfalls experienced in randomized optimization algorithms and neglect the ones already described by Johnson (2002). Experimentation on heuristics and metaheuristics got a wakeup-call by Hooker (1996), who criticized purely competitive testing as it was—and often still is—fairly common in metaheuristics papers. Instead, one should do scientific testing and investigate the effects which lead to high or low performance. These factors may be hidden in the algorithms, the problems, or the overall experimental setup. It is therefore most important for any experimental analysis on randomized optimization algorithms to set up a proper goal for the investigation (Sect. 14.3). This done, there are still several possibilities to fail. In the following, we list the most popular ones.

2.6.1 Floor and Ceiling Effects

In a paper known as the Rosenberg study (Gregory et al. 1996), the consequences of a floor effect on the outcome of an experimental work was impressively demonstrated. Two algorithms, a very simple one and a more sophisticated one, were compared concerning their ability to balance the load on a ring of processing elements. A large number of incoming jobs had to be distributed in order to maximize the throughput of the whole system. The expectation was that the more sophisticated scheme would obtain a clear advantage, but the results indicated that this was not the case. Both algorithms performed almost equally well. Further analysis revealed that the test case had been too hard: The number of incoming jobs was so high that neither algorithm could do very well. All processing elements were busy all the time, so that no improvement was possible. Thus, the test case was too hard to measure the expected effect. Depending on the perspective, this may also be interpreted as a ceiling effect. The trivial task of putting all processing elements to work was too easy to show any performance differences between the algorithms.

Both effects render an experimental study worthless and should be avoided. If observed in the results of a first experiment, the task has to be adjusted and the experiment repeated. In the context of optimization algorithms, a floor effect means that the contestants largely fail to attain the required task. A ceiling effect happens if they nearly always hit the required task, again with not much difference between the compared algorithms. A result table with all success rates near 100% would be a frequently observed example case.

2.6.2 Confounded Effects

When adapting flexible algorithms such as metaheuristics towards a problem or benchmark set, one is often tempted to change many operators or parameters at once because they are appealing to intuition after the first results are in. However, it is important to experimentally analyze not only the combination of new components but also the single effects of every change. Otherwise many unnecessary modifications may enter the algorithm despite the lack of evidence of their positive effect. In the worst case, it may happen that one of two newly incorporated operators is advantageous, while the second is disadvantageous, and the overall result is only slightly improved. Thus we recommend to do the following in the case of multiple algorithm modifications:

- Compare each change on its own with the default and the combined method. In case of more than two new additions, a factorial design (Chap. 3) may be useful.
- Document all the results at least in short whenever another researcher could possibly learn from this, even for cases which are not successful.

2.6.3 Fairness in Parameter Settings

When comparing different parametrizable algorithms, the employed parameter settings are extremely important as they largely define the obtained performance. Depending on the availability of code for the algorithms under scope and time for parameter searches, there are different possibilities to make a fair comparison:

- In the best case, the code for all methods is available. It is then possible to perform a parameter search for each problem and each algorithm via a tuning method (e.g., SPOT). Taking the best parameter sets for each method for each problem ensures comparing the algorithms at their peak performance.
- If algorithm runs on the chosen problems take too long for a full tuning process, one may however perform a simple space-filling design on the parameter space, e.g. a *Latin hypercube design* (LHD) with only a few design points and repeats. This prevents misconfigurations of algorithms as one probably easily gets into the "ball park" (De Jong 2007) of relatively good parameter settings.

Most likely, neither algorithm works at its peak performance level, but the comparison is still fair.

- If no code other than for one's own algorithm is available, one has to resort to comparing with default parameter values. For a new algorithm, these could be determined by a common tuning process over the whole problem set. Note however, that such comparison deliberately abstains from setting good parameters for specific problems, even if this would be attempted for any real-world application.

It goes without saying that comparisons of tuned versus untuned algorithms are not fair and should be avoided. This also applies to manual (one factor at a time) tuning. However, several tuning methods are available, including but not limited to REVAC (Chap. 12), the `F-Race` (Chap. 13), and SPOT (Chap. 14), all documented in this book. As the tuning (parameter optimization) problem poses a nondeterministic objective function without even approximate knowledge of global optimal performance, the question of the effort necessary (in algorithm runs) to "solve" it is difficult to answer. However, we advocate the use of fair conditions for the tuning as for the algorithm runs themselves and suggest an affordable medium-sized budget (for benchmark problems, 10^3 is an often used value)—the same for all algorithms.

2.6.4 Failure Reasons and Prevention

There are certainly many ways to fail when performing an experimentally supported investigation. One of these occurs when the original scientific goal—not the specific task in a derived experimental setup—cannot be reached because the experiments reveal that it is much harder to attain than expected, or not possible at all. This resembles a large-scale floor effect without the possibility to be cured easily as refining the overall task gradually can be very difficult. Another type of failure would happen if the assumptions about how things work or how components of algorithms combine are actually fundamentally wrong. The overall goal may be reachable, but not with the approach or techniques chosen.

Failures are surely unavoidable in experimental research, as it is the heart of scientific investigations that previously unexplored areas are tackled. However, it is specific to metaheuristics or other randomized algorithms that one has to cope with a lack of theory for many questions of practical importance. This puts an additional weight on the choice of the original research question that one tries to answer with an investigation. With the obtained negative results at hand, one may either try to refine the research question into one that is still interesting but easier to answer, or decide to learn as much as possible from the experiments and publish a well-founded negative paper. The following hints may be useful for avoiding failures:

- Experimentation should start as early as possible, even if not all features of the desired software are available yet. This reduces the danger of a complete failure and is well in line with recommendations in other sciences that strongly rely on experiments, such as those given by Thomke (2003) for innovation research in economics.

- Avoid watching a running experiment, and especially iterated manual parameter tuning or algorithm refinements. The impression obtained from seeing a few runs only can be highly misleading. It is therefore advisable to start interpreting the data only after the experiment is finished.
- Instead of only looking at the final summarizing performance values, it is highly recommendable to visualize what happens during the runs as well as to review the final results from different perspectives (e.g., multiple performance measures, several single runs, deviations from normal behavior, result distributions) to obtain as much insight as possible into the algorithm behavior.
- It is neither feasible nor desirable to beat the state-of-the-art algorithms on their own terrain by inventing a new algorithm in every publication, as is often attempted in metaheuristics. Instead, it may be much more rewarding to determine which algorithm is good on what kind of problem and to establish some guidelines for generalizing existing performance results to unknown problems or at least problem instances.
- Even for established algorithms, many interesting questions remain open which can be tackled only experimentally. Finding out more about the inner mechanisms of these algorithms may later help to refine them.

2.7 Tools: Measures and Reports

In the previous sections, we assumed that the experiment is done and may be investigated by means of various statistical techniques. However, given that we want to compare different algorithms or detect, e.g., the meaning of different parameter settings, several decisions have to be taken. The first is what to measure, and the others relate to the means of visually and textually displaying the results so that experimenter and audience benefit most from it.

2.7.1 Measures

When trying to outperform another algorithm which has been tested and documented on some benchmark functions, one often “inherits” the measure employed by other authors. However, considering the measure itself inevitably leads to the insight that much of the measuring process as exercised in stochastic optimization is fairly arbitrary.

A few measures are omnipresent in the stochastic optimization literature, possibly under different names, but with the same meaning. These are the *mean best fitness* (MBF), the *average evaluations to solution* (AES), the *success rate* (SR), and the *best-of-n* or *best ever*.

While the MBF is easy to apply, it has two shortcomings. Firstly, by taking the mean one loses all information about the result distribution, rendering a stable algorithm with mediocre performance similar to an unstable one coming up with very good solutions often, but also failing sometimes. Secondly, especially in situations where the global optimum is unknown, interpreting the obtained differences is difficult. Does an improvement of, e.g., 1% mean much, or not? In benchmarking, one usually observes a progression of the best solution on a logarithmic scale, evoked by the more and more exact approximation of the achieved optimum in later phases. In many practical settings, attaining a good local optimum itself is the hardest step and one would not be interested in infinitesimal improvements.

The latter situation may be better served by success rates or the AES as a performance measure. However, one has to set up meaningful quality values to avoid floor and ceiling effects. The AES holds more information as it measures the "time factor," whereas the SR may be easier to understand. However, the AES is ambiguous if the desired quality is not always reached. Auger and Hansen (2005) suggest the *success performance* measures SP1 and SP2, which deal with the unsuccessful runs in different ways depending on how they are stopped. The measures resemble a multistart extension of the AES and yield the number of function evaluations needed until the quality is finally reached, even if multiple runs are needed to get there.

The best-of-n value is most interesting in a design problem, as only the final best solution will be implemented, regardless of the number of runs n to get there. However, this measure strongly depends on n, rendering it unfeasible for comparisons on a benchmark level.

Whatever measure is used, one should be aware that it establishes only one specific cut through the algorithm output space. Bartz-Beielstein (2006) and Hansen et al. (2009) discuss this as the horizontal (fixed quality) and vertical (fixed time / evaluations) view. Every single measure can be misleading, e.g., if algorithm A_1 is good for short runs but algorithm A_2 usually overtakes it on long runs. Another set of conflicting criteria may be the performance on large problem instances against on small ones, see also (Eiben and Smith 2003). For the experimental analysis, the single measurement problem may be rectified by measuring several times on a horizontal scale, as suggested by Hansen et al. (2009). The obtained result is more accurate, but also more difficult to interpret. Nevertheless, rating algorithm performance with a single measured value needlessly oversimplifies the situation, at least in a benchmark environment where no external pressure enforces a specific measure.

However, if parameter tuning is considered, deriving an ordering of the tested algorithm designs cannot be avoided so that one has to select a specific performance measure. The tuning process should improve the algorithm performance according to the given measure, thus a strong influence of this measure on the resulting best algorithm design is highly likely. Whenever the standard measures do not deliver what is desired, one may be creative and design a measure that does, as suggested by Rardin and Uzsoy (2001).

2.7.2 Reporting Experiments

As indicated by the problems connected to experimental research we enumerated in the previous sections, setting up a good experiment is obviously not trivial. However, after conducting it, reporting on it constitutes another challenge. Article length requirements impose a limit on the level of possible detail. However, stating only results is useless; the description should rather deliver on three issues:

Replicability: One must be able to repeat the experiment, at least to a high degree of similarity.

Results: What is the outcome? Does it correspond to expectation? Are established differences statistically significant and scientifically meaningful?

Interpretation: Can the original research question be answered? What do the results mean regarding the algorithms, and the problems? Is it possible to generalize? Where do the observed effects originate from?

In contrast to in the natural sciences, there is no generally accepted scheme of how an experimental log should look in computer science. This may be partly due to the volatility of results. In a benchmark setting, it is technically easy and fast to run an experiment, and thinking about what to compare and how can take much longer. Additionally, there is no long tradition of empiricism in computer science. Many important works date from the 1980s and 1990s, e.g., McGeoch (1986), Sacks et al. (1989), and Barr et al. (1995).

However, for scientific readers as well as for the experimenters, a well-defined report structure could be beneficial: It provides standard guidelines for readers, what to expect, and where. Experimenters are regularly reminded to describe the important details needed to understand and possibly replicate their experiments. They are also urged to separate the outcome of fairly objective observing from subjective reasoning. Therefore, we propose organizing the presentation of experiments into seven parts, as follows:

ER-1: **Research question**
Briefly names the matter dealt with, the (possibly very general) objective, preferably in one sentence. This is used as the report's "headline" and related to the primary model.

ER-2: **Pre-experimental planning**
Summarizes the first—possibly explorative—program runs, leading to task and setup (ER-3 and ER-4). Decisions on employed benchmark problems or performance measures should be taken according to the data collected in preliminary runs. The report on pre-experimental planning should also include negative results, e.g., modifications to an algorithm that did not work or a test problem that turned out to be too hard, if they provide new insight.

ER-3: **Task**
Concretizes the question in focus and states scientific claims and derived statistical hypotheses to test. Note that one scientific claim may require several, sometimes hundreds, of statistical hypotheses. In case of a purely ex-

plorative study, as with the first test of a new algorithm, statistical tests may not be applicable. Still, the task should be formulated as precisely as possible. This step is related to the experimental model.

ER-4: **Setup**
Specifies problem design and algorithm design, including the investigated algorithm, the controllable and the fixed parameters, and the chosen performance measuring. The information provided in this part should be sufficient to replicate an experiment.

ER-5: **Results/Visualization**
Gives raw or produced (filtered) data on the experimental outcome and additionally provides basic visualizations where meaningful. This is related to the data model.

ER-6: **Observations**
Describes exceptions from the expected, or unusual patterns noticed, without subjective assessment or explanation. As an example, it may be worthwhile to look at parameter interactions. Additional visualizations may help to clarify what happens.

ER-7: **Discussion**
Decides about the hypotheses specified in ER-3, and provides necessarily subjective interpretations of the recorded observations. Also places the results in a wider context. The leading question here is: What did we learn?

In our view, it is especially important to divide parts ER-6 and ER-7, to facilitate different conclusions drawn by others, based on the same results/observations. This distinction into parts of increasing subjectiveness is similar to the suggestions of Barr et al. (1995), who distinguish between results, their analysis, and the conclusions drawn by the experimenter.

Note that all of these parts are already included in current good experimental reports. However, they are usually not separated but wildly mixed. Thus we only suggest to insert labels into the text to make the structure more obvious.

We also recommend to keep a journal of experiments with single reports according to the above scheme to enable referring to previous experiments later on. This is useful even if single experiments do not find their way into a publication, as it improves the overview of subsequent experiments and helps to avoid repeated tests.

2.8 Methodology, Open Issues, and Development

As detailed in this chapter, we can consider SPO as a methodological step forward for experimenting with parameterizable algorithms. Model building and tuning undoubtedly plays an important role here. However, there are some issues which currently remain less clear and should be tackled in the future. One of these is how to tune if runs take a very long time, so that only a few of them are affordable? This especially applies to many real-world applications, where tuning would probably greatly help but is rather difficult with the currently available methods. Another important issue concerns the amount of tuning to use: Where do we compare: After 500 algorithm runs, after 1000, or even more? Does this make a big difference at all? And in which cases? This is related to the adaptability of algorithms (Preuss 2009). Some algorithms may have a large tuning potential as they react sensitively to parameter changes, others largely stay at the default parameter performance. Both behaviors can be advantageous in different situations. The cited paper lays out only a first step in this direction by defining a measure for the *empirical tuning potential* (ETP). However, many methodological questions have to remain unresolved for now and invite further research.

2.9 Summary

Goals for the experimental analysis of computer algorithms were introduced in the first part of this chapter. Of great practical relevance is the improvement of an algorithm's performance and its robustness. Parameters of the algorithm were distinguished from problem parameters. This classification goes in line with the classification of the reasons why experiments are performed: Variation of the algorithm design can be applied in experimentation to improve, compare, or understand algorithm *efficiency* (tuning), whereas variation of the problem design might be helpful to analyze the *effectivity* (robustness) of an algorithm. Understanding of its working mechanism is another important task.

However, while performing experimental studies to accomplish these tasks, many problems can occur. We have considered problems related to the experimental setup, significance of the results, and building high-level theories. Models were considered as useful components for building a framework for experimentation. Starting from a discussion of model definitions in science, we claim that the representational view of models can be useful to bridge the gap between theory (linguistic objects, formulas) and experiment (computer programs, real-world optimization problems). These considerations are based on ideas presented by Giere (1999). A hierarchy consisting of primary, experimental, and data models was introduced. This hierarchy was developed in the framework of the new experimentalism, especially by Mayo (1996), who also developed tools for evaluating the scientific meaning of experimental results based on error statistics. SPO, as one possible framework to establish these goals, was introduced in this chapter.

In the later parts of the chapter, we discussed several practical issues of experimentation with randomized algorithms. Common pitfalls such as floor and ceiling effects and confounded effects were taken into account as well as fairness issues if applying tuning methods for comparing algorithms. We have highlighted some of the reasons why an experimental investigation can fail and then reviewed some important problem design issues: How do we measure, and how do we write up the results? Concerning the latter issue, we have introduced a seven-part result scheme that helps experimenters to structure their reports and also hinted at the use of an experimental journal. Additionally, we have named some still open problems, holes in the current methodology, which should be filled by future research. Concluding, we have shed some light on how experimentation might look in the future. Hopefully, correct statistics will be employed and correct conclusions drawn.

Acknowledgements This work has been supported by the Bundesministerium für Forschung und Bildung (BMBF) under the grant FIWA (AIF FKZ 17N2309, "Ingenieurnachwuchs") and by the Cologne University of Applied Sciences under the grant COSA.

References

Ackermann R (1989) The new experimentalism. British Journal for the Philosophy of Science 40:185–190

Auger A, Hansen N (2005) Performance Evaluation of an Advanced Local Search Evolutionary Algorithm. In: McKay B, et al. (eds) Proc. 2005 Congress on Evolutionary Computation (CEC'05), IEEE Press, Piscataway, NJ

Barr RS, Golden BL, Kelly JP, Resende MG, Stewart WR (1995) Designing and reporting on computational experiments with heuristic methods. Journal of Heuristics 1(1):9–32

Bartz-Beielstein T (2006) Experimental Research in Evolutionary Computation—The New Experimentalism. Natural Computing Series, Springer, Berlin, Heidelberg, New York

Bartz-Beielstein T (2008) How experimental algorithmics can benefit from Mayo's extensions to Neyman-Pearson theory of testing. Synthese 163(3):385–396, DOI 10.1007/s11229-007-9297-z

Beyer HG (2001) The Theory of Evolution Strategies. Springer

Chalmers AF (1999) What Is This Thing Called Science. University of Queensland Press, St. Lucia, Australia

Champion R (2009) What is this thing called falsificationism. `http://www.the-rathouse.com/shortreviews/WhatisThisThingCalledScience.html`. Cited 18 March 2009.

Cohen PR (1995) Empirical Methods for Artificial Intelligence. MIT Press, Cambridge MA

De Jong K (2007) Parameter Setting in EAs: a 30 Year Perspective. In: Fernando G Lobo and Cláudio F Lima and Zbigniew Michalewicz (ed) Parameter Setting in Evolutionary Algorithms, Springer

Demetrescu C, Italiano GF (2000) What do we learn from experimental algorithmics? In: MFCS '00: Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science, Springer, pp 36–51

Eiben AE, Smith JE (2003) Introduction to Evolutionary Computing. Springer

Fisher RA (1935) The Design of Experiments. Oliver and Boyd, Edinburgh

Giere RN (1999) Using models to represent reality. In: Magnani L (ed) Model Based Reasoning in Scientific Discovery. Proceedings of the International Conference on Model-Based Reasoning in Scientific Discovery, Kluwer, New York, NY, pp 41–57

Gregoire T (2001) Biometry in the 21st century: Whither statistical inference? (invited keynote). Proceedings of the Forest Biometry and Information Science Conference held at the University of Greenwich, June 2001. http://cms1.gre.ac.uk/conferences/iufro/proceedings/gregoire.pdf. Cited 19 May 2004

Gregory DE, Gao L, Rosenberg AL, Cohen PR (1996) An empirical study of dynamic scheduling on rings of processors. In: Proceedings of the 8th IEEE Symposium on Parallel and Distributed Processing, SPDP'96 (New Orleans, Louisiana, October 23-26, 1996), IEEE Computer Society, Los Alamitos, CA, pp 470–473

Hansen N, Auger A, Finck S, Ros R (2009) Real-parameter black-box optimization benchmarking 2009: Experimental setup. Tech. Rep. RR-6828, INRIA, URL http://hal.inria.fr/inria-00362649/en/

Hooker J (1996) Testing heuristics: We have it all wrong. Journal of Heuristics 1(1):33–42

Hoos HH, Stützle T (2005) Stochastic Local Search—Foundations and Applications. Elsevier/Morgan Kaufmann

Jägersküpper J, Preuss M (2008) Empirical investigation of simplified step-size control in metaheuristics with a view to theory. In: McGeoch CC (ed) Experimental Algorithms, 7th International Workshop, WEA 2008, Proceedings, Springer, Lecture Notes in Computer Science, vol 5038, pp 263–274

Johnson DS (2002) A theoretician's guide to the experimental analysis of algorithms. In: Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges, AMS, pp 215–250

Kleijnen JPC (1987) Statistical Tools for Simulation Practitioners. Marcel Dekker, New York, NY

Mayo DG (1983) An objective theory of statistical testing. Synthese 57:297–340

Mayo DG (1996) Error and the Growth of Experimental Knowledge. The University of Chicago Press, Chicago IL

Mayo DG, Spanos A (2006a) Severe testing as a basic concept in a neyman–pearson philosophy of induction. British Journal for the Philosophy of Science 57:323–357

Mayo DG, Spanos A (2006b) Severe Testing as a Basic Concept in a Neyman-Pearson Philosophy of Induction. British Journal Philos Sci 57(2):323–357, DOI 10.1093/bjps/axl003, URL http://bjps.oxfordjournals.org/cgi/content/abstract/57/2/323, http://bjps.oxfordjournals.org/cgi/reprint/57/2/323.pdf

Mayo DG, Spanos A (2010) Error and Inference. Cambridge University Press, Cambridge

McGeoch CC (1986) Experimental analysis of algorithms. PhD thesis, Carnegie Mellon University, Pittsburgh

Morrison DE, Henkel RE (eds) (1970) The Significance Test Controversy—A Reader. Butterworths, London, UK

Popper K (1959) The Logic of Scientific Discovery. Hutchinson, London, UK

Preuss M (2009) Adaptability of algorithms for real-valued optimization. In: Giacobini M, et al. (eds) Applications of Evolutionary Computing, EvoWorkshops 2009. Proceedings, Springer, Lecture Notes in Computer Science, vol 5484, pp 665–674

Rardin R, Uzsoy R (2001) Experimental evaluation of heuristic optimization algorithms: A tutorial. Journal of Heuristics 7(3):261–304

Sacks J, Welch WJ, Mitchell TJ, Wynn HP (1989) Design and analysis of computer experiments. Statistical Science 4(4):409–435

Schwefel HP (1995) Evolution and Optimum Seeking. Sixth-Generation Computer Technology, Wiley, New York, NY

Suppes P (1969) A comparison of the meaning and uses of models in mathematics and the empirical sciences. In: Suppes P (ed) Studies in the Methodology and Foundation of Science, Reidel, Dordrecht, The Netherlands, pp 11–13

Thomke SH (2003) Experimentation Matters: Unlocking the Potential of New Technologies for Innovation. Harvard Business School Press

Chapter 3
Design and Analysis of Computational Experiments: Overview

Jack P.C. Kleijnen

Abstract This chapter presents an overview of the design and analysis of computational experiments with optimization algorithms. It covers classic designs and their corresponding (meta)models; namely, Resolution-III designs including fractional factorial two-level designs for first-order polynomial models, Resolution-IV and Resolution-V designs for two-factor interactions, and designs including central composite designs for second-degree polynomials. It also reviews factor screening in experiments with very many factors, focusing on the sequential bifurcation method. Furthermore, it reviews Kriging models and their designs. Finally, it discusses experiments aimed at the optimization of the parameters of a given optimization algorithm, allowing multiple random experimental outputs. This optimization may use either generalized response surface methodology or Kriging combined with mathematical programming; the discussion also covers Taguchian robust optimization.

3.1 Introduction

There are many different optimization algorithms; e.g., there are several metaheuristics including evolutionary algorithms and the related genetic algorithms, *response surface methodology* (or RSM; see Section 3.5.1), simulated annealing, and tabu search; see the recent overviews by Adenso-Diaz and Laguna (2006), Bartz-Beielstein (2006), Kleijnen (2008), Myers and Montgomery (1995) and Rajagopalan et al. (2007).

Before users can apply such an algorithm, they must quantify specific parameters; e.g., for a particular evolutionary algorithm they must specify the number

Jack P.C. Kleijnen
Department of Information Management / CentER
Tilburg University, Postbox 90153, 5000 LE Tilburg, Netherlands.
`http://center.uvt.nl/staff/kleijnen`

of parent individuals and the recombination operator. To recommend optimal values for these parameters, optimization experts may try different combinations of parameter values. To select these combinations, they should use statistical techniques that have been developed for the design and analysis of experiments. These techniques are the focus of this chapter.

These techniques have been developed for the design and analysis of two types of experiments:

- general experiments
- computational experiments

These techniques are known in the literature as *design of experiments* (DOE) for the first type of experiments; see, e.g., Montgomery (2009), and as *design and analysis of computer experiments* (DACE) for the second type. DACE assumes that these computer codes are deterministic simulation models, as used in computer aided engineering and physics; see Santner et al. (2003). Kleijnen (2008) proposes the term *design and analysis of simulation experiments* (DASE) for experiments with either deterministic or random simulation. Deterministic simulation models give the same output whenever the combination of input values is the same. Random simulation models, however, use (pseudo)random numbers as input besides other inputs, so these models give outputs that vary with the random numbers used. These random models are used to study supply chains, telecommunication networks, etc.; see Law (2007). Real-life experiments also give random outputs because these outputs depend not only on the inputs that are controlled by the experimenters but also on uncontrollable environmental factors; also see the Taguchian worldview in Sect. 3.5.3. Note that all experimental designs treat the experiment as a *black box*; i.e., only *input/output* (I/O) data are observed. Bartz-Beielstein (2006) presents examples of the use of these statistical techniques for experiments with evolutionary computation; more references to such experiments follow below.

Whereas in real-life experimentation it is not practical to investigate many factors (inputs), experiments with optimization algorithms may have hundreds of factors—a *factor* may be a parameter of the optimization algorithm or the particular optimization problem (e.g., the traveling salesperson problem) to be solved by that optimization algorithm; examples of such parameters are the problem's size and structure. Moreover, whereas in real-life experiments it is hard to vary a factor over many values, in optimization algorithms this restriction does not apply, e.g., *Latin hypercube sampling* (LHS) is a design that has as many values per factor as it has combinations; see Sect. 3.4. Consequently, a multitude of *scenarios*—combinations of factor values—may be observed. Moreover, these experiments are well-suited to the application of sequential designs instead of one-shot designs (ignoring parallel computers); see the chapters in this book by Ridge and Kudenko (2010) and Bartz-Beielstein and Preuss (2010). So a change of mindset of the experimenter is necessary, as Kleijnen et al. (2005) pointed out.

Design and *analysis* of experiments are intertwined (chicken and egg). The analysis uses—implicitly or explicitly—a *metamodel* (also called response surface, surrogate, emulator, etc.), which is an approximation of the I/O function of the experiment; i.e., the experiment yields I/O data that are used to estimate this function. There are different *types* of metamodels. The most popular type are polynomials of first or second order (degree); see Sect. 3.2. Another metamodel type that is becoming popular is Kriging (also called the Gaussian Process model); see Sect. 3.4. A less popular type is the *generalized linear model* (GLM); see the discussion by Bartz-Beielstein (2006). References to many other types are given by Kleijnen (2008, p. 8).

Each type of metamodel requires a different design type; e.g., a first-order polynomial may be estimated (fitted) through a Resolution-III design, as we shall see in Sect. 3.2, whereas a Kriging model may be estimated through a space-filling design such as provided by LHS. The adequacy of the design and metamodel combination depends on the *goal* of the experiment. For example, if the goal is to estimate gradients (or derivatives), then a first-order polynomial may be adequate. However, if the goal is global approximation, then Kriging may be better. The role of different goals is further discussed by Kleijnen and Sargent (2000). Experiments with optimization algorithms have two main goals:

- Sensitivity analysis
- Optimization

Sensitivity analysis may serve *factor screening*—or briefly screening—which denotes the search for the most important factors among the many factors that are varied in an experiment; also see Ridge and Kudenko (2010). Optimization tries to find the optimal combination of the optimization algorithm parameters (optimization may follow after sensitivity analysis). *Robust* optimization allows for *uncertain* environmental factors, which depend on the problems to be solved and the particular optimization algorithm.

We now present a detailed overview of the remainder of this chapter, so readers can decide whether they wish to skip specific sections.

Section 3.2 covers classic designs and their corresponding metamodels. Resolution-III (R-III) designs assume first-order polynomial metamodels; these design include so-called Plackett–Burman and 2^{k-p} designs. Resolution-IV (R-IV) and Resolution-V (R-V) designs assume important two-factor interactions. Designs for second-degree polynomials include *central composite designs* (CCDs). Note that, compared with these designs, the traditional approach of changing only one factor at a time is inferior; the latter approach is called the finite difference method in gradient estimation; see Kleijnen (2008, pp. 8, 29, 32-34, 107, 118), and also Fu (2008).

Section 3.3 reviews screening, focusing on sequential bifurcation. Traditionally, experimenters use prior knowledge to select a few factors, simply assuming that these factors are the most important ones. In a recent case study with 92 factors, Kleijnen et al. (2006a) applied sequential bifurcation and identified a shortlist with 10 factors after investigating only 19 combinations.

Section 3.4 reviews Kriging and its designs. Kriging models are fitted to data that are obtained for larger experimental areas than the areas used in low-order polynomial regression; i.e., Kriging models are *global* rather than local. Kriging is used for prediction; its final goals are sensitivity analysis and optimization.

Section 3.5 discusses experiments aimed at optimization, focusing on experiments with multiple random outputs and assuming that one output is selected as the goal (objective) output while the other random outputs must satisfy prespecified target values (thresholds). This section covers two methods, namely *generalized response surface methodology* (GRSM) which fits a series of local low-order polynomials, and Kriging combined with mathematical programming. This section also considers robust optimization in the sense of Taguchi. Note that the methods in Sect. 3.5 are specific types of optimization algorithms; other types have already been mentioned at the very beginning of this section.

3.2 Classic Designs and Metamodels

Classic designs and their corresponding metamodels are detailed in many DOE textbooks such as Montgomery (2009) and Myers and Montgomery (1995); DOE for simulation experiments is detailed by Kleijnen (2008).

Let us assume a *first-order polynomial* metamodel for an experiment with k factors:

$$E(y) = \beta_0 + \beta_1 x_1 + \ldots + \beta_k x_k, \tag{3.1}$$

where $E(y)$ denotes the expected value (mean) of the metamodel's output y; β_0 is called the intercept, and β_j $(j = 1, \ldots, k)$ is called the main effect or first-order effect of input x_j. To estimate the effects $\beta = (\beta_0, \beta_1, \ldots, \beta_k)'$ we obviously need at least $k + 1$ combinations. Table 3.1 presents a 2^{7-4}_{III} design, which is to be read as follows. The columns with the symbols **1** through **7** give the values of the corresponding factor in the experiment; $\mathbf{4} = \mathbf{1.2}$ means that the value of factor 4 equals the product of the values of the factors 1 and 2 in the corresponding combination; the symbol $-$ means that the factor has the standardized value -1 (which corresponds with the lowest value in the original scale) and $+$ means that the factor has the standardized value $+1$ (highest original value). This design has Resolution III, which means that it enables unbiased estimation of the coefficients of the first-order polynomial—assuming such a polynomial is an adequate approximation of the I/O function. This particular design investigates only $2^{7-4} = 8$ combinations; i.e., it investigates only a fraction $2^{-4} = 1/16$ of all $2^7 = 128$ combinations of a full two-level factorial design. Note that each column in Table 3.1 is *balanced*; i.e., it has an equal number of $+$ and $-$ signs; moreover, all columns are *orthogonal*.

A R-III design enables *ordinary least squares* (OLS) estimation of the first-order effects β_j and the intercept β_0 in (3.1). The OLS estimator is

$$\hat{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{w}, \tag{3.2}$$

where $\mathbf{X}$ is the $n \times q$ matrix with the explanatory variables in the regression model where the intercept corresponds with a column that has only the value $+1$ (e.g.,

Table 3.1: A $1/16$ fractional two-level factorial design for seven factors

Combi.	$\mathbf{1}$	$\mathbf{2}$	$\mathbf{3}$	$\mathbf{4 = 1.2}$	$\mathbf{5 = 1.3}$	$\mathbf{6 = 2.3}$	$\mathbf{7 = 1.2.3}$
1	$-$	$-$	$-$	$+$	$+$	$+$	$-$
2	$+$	$-$	$-$	$-$	$-$	$+$	+
3	$-$	$+$	$-$	$-$	$+$	$-$	+
4	$+$	$+$	$-$	$+$	$-$	$-$	$-$
5	$-$	$-$	$+$	$+$	$-$	$-$	+
6	$+$	$-$	$+$	$-$	$+$	$-$	$-$
7	$-$	$+$	$+$	$-$	$-$	$+$	$-$
8	$+$	$+$	$+$	$+$	$+$	$+$	+

in the 2^{7-4} example $n = 8$ and $q = 1 + 7 = 8$) and $\mathbf{w}$ is the vector with the n experimental outputs. Because Table 3.1 gives an orthogonal design, the OLS estimator simplifies to

$$\widehat{\beta_j} = \sum_{i=1}^{8} x_{i;j} w_i / 8 \qquad (j = 0, 1, \ldots, 7).$$

OLS is the classic estimation method in linear regression analysis, assuming *white noise*; i.e., the regression residuals are *normally* (Gaussian), *independently, and identically distributed* (NIID) with zero mean (we shall return to this assumption at the end of this section).

Using *standardized* values of -1 and $+1$ implies that the estimated main effects quantify the *relative* importance of the k inputs; e.g., the most important input is the one that gives $\max_{1 \leq j \leq k} |\widehat{\beta_j}|$. So sensitivity analysis should use coded input values. Optimization, however, should use the original values (unfortunately, an optimization method such as RSM that uses steepest descent gives scale-dependent results; also see Sect. 3.5.1). Anyhow, each input combination (specified by the row of a design matrix such as Table 3.1) must be translated into the original values before the experiment can start. Standardization is further discussed by Kleijnen (2008, pp. 30–31).

Table 3.1 displays a design with only $2^{7-4} = 8$ combinations, which is an attractive design if the first-order polynomial is an adequate approximation and the white-noise assumption holds. However, if higher-order effects are important, then this reduction in the number of combinations makes the estimated effects biased. Therefore we present several other types of design.

A R-IV design ensures that the estimated first-order effects $\widehat{\beta}$ are not biased (confounded, aliased) by the two-factor interactions $\beta_{j;j'}$ $(j < j' = 2, \ldots k)$; these interactions imply that the $k(k-1)/2$ terms $\beta_{j;j'} xj x_{j'}$ are added to (3.1). A R-IV design is easily constructed from a R-III design: if $\mathbf{D}$ denotes the R-III design, then add $-\mathbf{D}$ (the negative or foldover of $\mathbf{D}$) to $\mathbf{D}$ (so the R-IV design doubles the number of combinations). For example, the R-IV design that is based on Table 3.1 has only minuses in its last row (row 16), because this row is the negative of row 8 in Table 3.1.

To estimate the $k(k-1)/2$ individual interactions, a R-V design is needed. An example of such a R-V design for $k = 7$ factors is a 2^{7-1} design; however, its 64 combinations take too much computer time if the experiment with the optimization algorithm is computationally expensive. In that case, it is better to use a *saturated* design, which by definition has a number of combinations n equal to the number of metamodel parameters q (e.g., $q = 1 + 7 + 21 = 29$). Such saturated R-V designs are Rechtschaffner's designs, which are reproduced by Kleijnen (2008, p. 49).

If the experimenter assumes a *second-degree polynomial*, then a CCD also enables the estimation of the k purely quadratic effects $\beta_{j;j}$ $(j = 1, \ldots, k)$. Such a CCD augments the R-V design with the central point of the experimental area and $2k$ axial points, which change each factor one-at-a-time by $-c$ and c units where $c > 0$. Obviously the CCD is rather wasteful in case of expensive experiments, because it has five values per factor if $c \neq 1$ (instead of the minimum, three) and it is not saturated. Alternatives for the CCD are discussed by Kleijnen (2008) and Myers and Montgomery (1995).

The various classic designs can be used *sequentially*. So, we may start with a R-III design and test its adequacy. There are various tests; e.g., *cross-validation* deletes I/O combination i $(i = 1, \ldots, n)$, and computes the corresponding estimator from the analogue of (3.2):

$$\hat{\beta}_{-i} = (\mathbf{X}'_{-i}\mathbf{X}_{-i})^{-1}\mathbf{X}'_{-i}\mathbf{w}_{-i}, \tag{3.3}$$

where we assume that the inverse does exist (a necessary condition is $n > q$). If the re-estimated effects $\hat{\beta}_{-i}$ and the original effects $\hat{\beta}$ differ greatly, then the first-order polynomial is not an adequate approximation. We can then proceed to a R-IV design, adding the negative of the R-III design**.** If the R-IV design gives very different estimated first-order effects, then two-factor interactions may be important so we proceed to a R-V design (unfortunately, the R-IV design is not always a proper subset of the R-V design). If the model with two-factor interactions turns out not to be adequate, then we may proceed to a CCD (which does have the R-V design as a subset).

To select a specific design, users may learn how to construct experimental designs. An alternative is to use software or websites. Textbooks, software, and websites are given by Kleijnen (2008, pp. 51, 105, 130).

An application of these classic designs is provided by Bartz-Beielstein (2006). He starts with $k = 9$ inputs. The output variable w is the quality of the best solution found by the search strategy. He uses a 2^{9-5}_{III} design, to find the important factors; he transforms the output w into $\log(w)$. He then finds that only five of these nine factors are significant. Next, for the four significant quantitative factors, he uses a full factorial two-level design (or 2^4 design). The resulting I/O data give only two important factors. He explores these two factors in a CCD design.

The assumptions of these classic designs and metamodels stipulate univariate output (responses) and white noise. Kleijnen (2008) discusses multivariate (multiple) outputs, nonnormality of the output (solved through the statistical techniques called jackknifing and bootstrapping), variance heterogeneity (different input combinations give different output variances), and *common random numbers* (CRN) (solved through either adapted OLS or generalized least squares), and testing the validity of low-order polynomial metamodels (through either cross-validation or the F lack-of-fit statistic). Note that the goal of CRN is to reduce the variance of the estimated factor effects; i.e., CRN uses the same pseudorandom numbers to obtain one observation (or replicate) for each of the n scenarios so the n simulation outputs are positively correlated (not independent; if each of the n scenarios is simulated $m > 1$ times using different pseudorandom numbers in different replicates, then these m replicates are independent); see Law (2007, pp. 578–594).

3.3 Screening: Sequential Bifurcation

R-III designs (discussed in Sect. 3.2) are called screening designs by some authors (e.g., Ridge and Kudenko (2007), Yu (2007)), but we reserve the term screening for experiments with so many factors that a R-III design with its $n > k$ combinations takes too much computer time. Screening is related to phenomena such as sparse effects, parsimony, Pareto's principle, Occam's razor, 20–80 rule, and curse of dimensionality. Practitioners do not yet apply screening methods; instead, they experiment with a few intuitively selected factors only. The following two case studies illustrate the need for screening.

Bettonvil and Kleijnen (1996) present a greenhouse deterministic simulation model with 281 factors. Politicians wanted to take measures to reduce the release of CO_2 gasses; they recognized that they should start with legislation for a few factors only. The authors managed to find the 15 most important factors after simulating 154 combinations of the 281 factors, accounting for two-factor interactions.

Another case study is presented by Kleijnen et al. (2006a). This case concerns a random simulation of a supply chain centered around the Ericsson company in Sweden. This simulation has 92 factors. The authors use sequential bifurcation to identify a shortlist with 10 factors, after simulating only 19 combinations.

There are several types of screening designs; see the references in Kleijnen (2008, p. 158–159). Sequential bifurcation is the most efficient and effective design if its assumptions are indeed satisfied; also see Xu et al. (2007). For its design, sequential bifurcation uses the following metamodel *assumptions*:

1. A first-order polynomial augmented with two-factor interactions is an adequate metamodel (approximation).
2. All first-order effects have known signs and are nonnegative.
3. There is strong heredity; i.e., if a factor has no important main effect, then this factor does not interact with any other factor; also see Wu and Hamada (2000).

The role of these assumptions becomes clearer when we consider the sequential bifurcation procedure in some more detail. The first step aggregates all factors into a single group, and changes the value of that group of factors from -1—that is, all individual factors within that group have the value -1—to $+1$—so all individual factors within that group have the value $+1$. No cancellation of individual effects occurs, given the assumptions 1 and 2. If that group indeed has a significant effect—which is most likely in the first step—then the second step splits the group into two subgroups—it bifurcates—and tests each of these subgroups for importance. In the next steps, sequential bifurcation splits important subgroups into smaller subgroups, and keeps unimportant subgroups unchanged. In the final step, all individual factors that are not in subgroups identified as nonsignificant are estimated and tested.

This procedure may be interpreted through the following *metaphor*. Imagine a lake that is controlled by a dam. The goal of the experiment is to identify the highest (most important) rocks; actually, sequential bifurcation not only identifies but also measures the height of these rocks. The dam is controlled in such a way that the level of the murky water slowly drops. Obviously, the highest rock emerges from the water first! The most-important-but-one rock turns up next, etc. Sequential bifurcation stops when the analysts feel that all the important factors are identified; once sequential bifurcation stops, the analysts know that all remaining (unidentified) factors have smaller effects than the effects of the factors that have been identified. Obviously, this property is important for practice.

There is a need for more research in the following areas:

- It is a challenge to derive the number of replicates that control the overall probability of correctly classifying the individual factors as important or unimportant (currently, sequential bifurcation applies a statistical test to each subgroup individually).
- After sequential bifurcation stops, the resulting shortlist of important factors should be validated (never trust a metamodel).
- Multiple performance measures (instead of single one) of the optimization algorithm may require different bifurcation patterns; this complication deserves more research.
- There is a need to develop software that implements sequential bifurcation.
- A contest may be organized to challenge different screening methods to find the important parameters of different optimization algorithms (e.g., evolutionary algorithms) for problems (e.g., traveling salesperson). Testbeds with different optimization problems are popular in mathematical programming.

3.4 Kriging

Kriging was originally developed in geostatistics—also known as spatial statistics—by the South African mining engineer Danie Krige. A classic geostatistics textbook is that by Cressie (1993). Later on, Kriging was applied to the I/O data of deterministic simulation models; see Sacks et al. (1989) and also Santner et al. (2003). Only recently Van Beers and Kleijnen (2003) applied Kriging to random simulation models; Kriging in random simulation is also investigated by Ankenman et al. (2009) and Yin et al. (2008). The track record of Kriging in deterministic simulation holds great promise for Kriging in other domains including optimization algorithms.

The simplest type of Kriging is called *ordinary* Kriging, and assumes

$$w(\mathbf{d}) = \mu + \delta(\mathbf{d}), \tag{3.4}$$

where $w(\mathbf{d})$ denotes the experimental output for input combination $\mathbf{d}$, μ is the output averaged over the whole experimental area, and $\delta(\mathbf{d})$ is the additive noise that forms a stationary covariance process (instead of the white noise in regression analysis, which implies zero covariances). This Kriging uses the following *linear* predictor:

$$y(\mathbf{d}) = \lambda' \mathbf{w}, \tag{3.5}$$

where the weights λ are not constants—whereas the regression parameters β are—but decrease with the *distance* between the new input $\mathbf{d}$ to be predicted and the old points collected in the $n \times k$ design matrix $\mathbf{D}$ where n denotes the number of input combinations and k the number of inputs (like in Sect. 3.2). The *optimal* weights can be proven to be

$$\lambda_o = \mathbf{\Gamma}^{-1}[\gamma + \mathbf{1}\frac{1 - \mathbf{1}'\mathbf{\Gamma}^{-1}\gamma}{\mathbf{1}'\mathbf{\Gamma}^{-1}\mathbf{1}}], \tag{3.6}$$

where $\mathbf{\Gamma} = (cov(w_i, w_{i'}))$ with $i, i' = 1, \ldots, n$ is the $n \times n$ matrix of the covariances between the old outputs; $\gamma = (cov(w_i, w_0))$ is the n-dimensional vector of the covariances between the n old outputs w_i and w_0, the output of the combination to be predicted, which may be either new or old. Actually, (3.4), (3.5), and (3.6) imply that the predictor may be written as

$$y(\mathbf{d}) = \widehat{\mu} + \gamma(\mathbf{d})'\mathbf{\Gamma}^{-1}(\mathbf{w} - \widehat{\mu}\mathbf{1}), \tag{3.7}$$

where

$$\widehat{\mu} = (\mathbf{1}'\mathbf{\Gamma}^{-1}\mathbf{1})^{-1}\mathbf{1}'\mathbf{\Gamma}^{-1}\mathbf{w}.$$

The *gradient* follows from (3.7); an example is given in Kleijnen (2008, pp. 143, 145).

The covariances in $\mathbf{\Gamma}$ and γ are often based on the *correlation function*

$$\rho = \exp[-\sum_{j=1}^{k} \theta_j h_j^{p_j}] = \prod\nolimits_{j=1}^{k} \exp[-\theta_j h_j^{p_j}], \tag{3.8}$$

where h_j denotes the distance between the input d_j of the new and the old combinations, θ_j denotes the importance of input j (the higher θ_j is, the less effect input j has), and p_j denotes the smoothness of the correlation function (e.g., $p_j = 2$ implies an infinitely differentiable function). So-called exponential and Gaussian correlation functions have $p = 1$ and $p = 2$, respectively. Obviously, the correlation function (3.8) implies that the weights are relatively high for inputs close to the input to be predicted. Furthermore, we found experimentally that some of the optimal weights in (3.6) may be negative; see Table 3.2. Finally, the weights imply that for an old input the predictor equals the observed output at that input:

Table 3.2: A Kriging example

Combi.	x	w	$\lambda(0.57)$
1	0.3	0.429	0.024
2	0.4	0.667	−0.127
3	0.5	1.000	0.433
4	0.6	1.500	0.720
5	0.7	2.333	−0.051

$$y(\mathbf{d}_i) = w(\mathbf{d}_i) \text{ if } \mathbf{d}_i \in \mathbf{D}, \tag{3.9}$$

so all weights are zero except the weight of the observed output; i.e., the Kriging predictor is an *exact interpolator* (the OLS regression predictor minimizes the sum of squared residuals, so it is not an exact interpolator, unless $n = q$).

The interpolation property (3.9) is attractive in *deterministic* simulation, because the observed simulation output is unambiguous. In experiments with *random outputs*, however, the observed output is only one of the many possible values. For random simulations, Van Beers and Kleijnen (2003) replace $w(\mathbf{d}_i)$ by the average observed output $\overline{w_i}$. Those authors give examples for which the Kriging predictions are more accurate than the regression predictions (regression metamodels may be useful for other goals, e.g., understanding, screening, and validation).

We present a Kriging example in Table 3.2. The true I/O function of this one-dimensional ($k = 1$) experiment is $w = x/(1 - x)$ (which resembles the I/O function of a single-server queueing model). We select $n = 5$ equidistant input values in the experimental area $[0.3, 0.7]$. Note that these values are given by either a space-filling design or a CCD with $c = 0.1$. Suppose we wish to predict the output for the input $x = 0.57$. We display the Kriging weights λ of the five old input values for this new input value, which imply that higher weights are given to output values of input values nearby (the sum is 1, ignoring rounding errors). The resulting prediction error is $w - \widehat{y}$ = 1.326 − 1.321 = 0.005. Fitting a second-order polynomial (see Sect. 3.2) would have given $\widehat{\beta}$ = $(1.094, -5.051, 9.694)'$, so the regression predictor for $x = 0.57$ would have been $(1.094, -5.051,' 9.694)'(1, 0.57, 0.57^2)$ = 1.364 and the resulting prediction error would have been $1.326 - 1.364 = -0.038$, which exceeds the Kriging prediction error (0.005).

3.4.1 The Kriging Predictor Variance

A major problem in Kriging is that *the correlation function is unknown*. Therefore both the type and the parameter values must be estimated. To estimate the parameters, the standard Kriging literature and software uses *maximum likelihood estimators*(MLEs). The estimation of the correlation functions and the corresponding optimal weights in (3.6) can be done through the Matlab toolbox called DACE, which is software that is well documented and free of charge; see Lophaven et al. (2002).

The Kriging literature virtually ignores problems caused by replacing the optimal weights λ in (3.5) by the estimated optimal weights (say) $\widehat{\lambda_0}$—using the estimated correlation parameters $\widehat{\theta_j}$; see (3.8) and the references to the Kriging literature given above. Actually, this replacement makes the Kriging predictor a *nonlinear* estimator. The literature uses the predictor variance—*given* the Kriging weights λ. This implies a zero variance in case the new point w_0 equals an old point w_i; also see (3.9). Furthermore this tends to underestimate the true variance. Finally, this variance and the true variance do not reach their maxima for the same input combination, which is important in sequential designs; see Sect. 3.4.2 below. More details are given by Den Hertog et al. (2006), assuming the simulation models are deterministic (so those authors apply parametric bootstrapping).

In random simulation, each input combination is replicated a number of times so a simple method for estimating the true predictor variance and the true gradient is *distribution-free bootstrapping*. The basics of bootstrapping are explained in Efron and Tibshirani (1993) and Kleijnen (2008). To estimate the predictor variance, Van Beers and Kleijnen (2008) resample—with replacement—the (say) m_i replicates for combination i $(i = 1, \ldots, n)$. This sampling results in the bootstrapped average $\overline{w_i^*}$ where the superscript asterisk is the usual symbol to denote a bootstrapped observation. These n bootstrapped averages $\overline{w_i^*}$ give estimated correlation coefficients $\widehat{\theta_j}^*$, estimated optimal weights $\widehat{\lambda_0^*}$, and the Kriging predictor y^*. To decrease sampling effects, this whole procedure is repeated B times (e.g., B = 100), which gives y_b^* with $b = 1, \ldots, B$. The variance of the Kriging predictor is estimated from these B values.

3.4.2 Designs for Kriging

To get the I/O data to which the Kriging model is fitted, experimenters often use LHS. This design assumes that a valid metamodel is more complicated than a low-order polynomial (assumed by classic designs). LHS does not assume a specific metamodel; instead, LHS focuses on the design space formed by the k-dimensional cube defined by the k standardized inputs. LHS is one of the space-filling types of design; another space-filling design type is a max–min design, which maximizes the minimum distance between the n input combinations. Space-filling designs and software for obtaining these designs are discussed by Kleijnen (2008).

Instead of a one-shot space-filling design such as a LHS design, a *sequentialized* design may be used. In general, sequential statistical procedures are known to require fewer observations than fixed-sample (one-shot) procedures; also see Park et al. (2002). Sequential designs imply that observations are analyzed—so the data generating process is better understood—before the next input combination is selected. This property implies that the design depends on the specific underlying process (the optimization algorithm and the problem to be solved by the optimization algorithm); i.e., the design is customized (tailored or application driven, not generic). Moreover, computational experiments (unlike real-life experiments) proceed sequentially. The following sequential design for Kriging in sensitivity analysis is developed by Van Beers and Kleijnen (2008):

1. Start with a *pilot* experiment, using some small generic space-filling design; e.g., a LHS or a max–min design.
2. Fit a *Kriging* model to the I/O data that are available at this step (in the first pass of this procedure, these I/O data are the data resulting from Step 1).
3. Consider (but do not yet simulate) a set of *candidate* input combinations that have not yet been observed and that are selected through some space-filling design; select as the next combination to be actually run, the candidate combination that has the *highest predictor variance* (this variance may be estimated through bootstrapping, as we saw above).
4. Use the combination selected in Step 3 as the input combination to be run (expensive!), and obtain the corresponding output data.
5. Re-fit a Kriging model to the I/O data that is augmented with the I/O data resulting from Step 4.
6. Return to Step 3, until the Kriging metamodel is acceptable for its goal (sensitivity analysis) and stop.

The resulting design is indeed *customized*; i.e., which combination has the highest predictor variance is determined by the underlying problem and optimization algorithm; e.g., if the true (unknown) I/O function is a simple hyperplane within a subspace of the total experimental area, then this design selects relatively few points in that part of the input space. A sequential design for constrained optimization (instead of sensitivity analysis) will be presented in Sect. 3.5.2.

There is a need for more research on Kriging, in the following areas:

- Kriging *software* needs further improvement; e.g., Kriging should allow random outputs with variance heterogeneity and correlation created by CRN.
- Sequential designs may benefit from *asymptotic proofs* of their performance; e.g., does the design approximate the optimal design?
- More experimentation and analyses may be done to derive *rules of thumb* for the sequential design's parameters, such as the size of the pilot design and the initial number of replicates.
- *Stopping rules* for sequential designs based on a measure of accuracy may be investigated.
- Nearly all Kriging publications assume univariate output, whereas in practice computational experiments have *multivariate output*.
- Often the analysts know that the experiment's I/O function has certain properties, e.g., monotonicity. Most metamodels (such as Kriging and regression) do not preserve these properties. A new property-preserving Kriging method has been developed by Kleijnen and Van Beers (2009).

3.5 Optimization

As we have already mentioned in the very first sentence of Sect. 3.1, there are many optimization algorithms. In the present section we shall discuss optimization algorithms that use designs and metamodels that we have also presented in the preceding sections. In Sect. 3.5.1 we review RSM, which uses classic designs and low-order polynomials discussed in Sect. 3.2. In Sect. 3.5.2 we survey Kriging (see Sect. 3.4) combined with mathematical programming. In Sect. 3.5.3 we survey robust optimization in the sense of Taguchi (1987), using either RSM or Kriging.

3.5.1 Response Surface Methodology (RSM)

Originally, Box and Wilson (1951) developed RSM for the optimization of real-life systems; also see Myers and Montgomery (1995). Later on, Angün et al. (2009) developed *generalized RSM* (GRSM), which allows multiple (multivariate) random responses; GRSM assumes that one response is the goal (objective) response and the other responses are constrained variables that must satisfy prespecified target values. Both GRSM and RSM estimate gradients to search for the optimum. These gradients are based on local first-order polynomial approximations; also see (3.1). GRSM combines these gradients with mathematical programming findings to estimate a better search direction than the steepest descent direction used by RSM. Moreover, GRSM uses these gradients in a bootstrap procedure for testing whether the estimated solution satisfies the *Karush–Kuhn–Tucker* (KKT) optimality conditions. Some details now follow.

Classic RSM has the following *characteristics*:

- RSM is an *optimization heuristic* that tries to estimate the input combination that minimizes a given univariate black-box goal function.
- RSM is a *stepwise* (multi stage) method.
- In these steps, RSM uses local first-order and second-order *polynomial* metamodels (response surfaces). RSM assumes that these models have *white noise* in the local experimental area; when moving to a new local area in a next step, the variance may change.
- To fit these first-order polynomials, RSM uses *classic R-III designs*; for second-order polynomials, RSM usually applies a classic CCD.
- To determine in which direction the inputs will be changed in a next step, RSM uses steepest descent based on the *gradient* implied by the first-order polynomial fitted in the current step: equation (3.1) implies the estimated gradient

$$\widehat{\beta}_{-0} = (\widehat{\beta}_1, \ldots, \widehat{\beta}_k)',$$

  where the subscript -0 means that the estimated intercept $\widehat{\beta}_0$ is deleted from $\widehat{\beta}$ defined in (3.2).

- In the final step, RSM takes the gradient of the locally fitted *second-order* polynomial to estimate the optimum input combination. RSM may also apply *canonical analysis* to examine the shape of the optimal (sub)region: unique minimum, saddle point, ridge?

Kleijnen et al. (2006b) derive a variant of steepest descent—called *adapted steepest descent*—$[\mathbf{cov}(\widehat{\beta_{-0}})]^{-1}\widehat{\beta}_{-0}$, where $\mathbf{cov}(\widehat{\beta_{-0}})$ is the matrix of the covariances between the components of the estimated gradient $\widehat{\beta_{-0}}$. Their variant gives a scale-independent search direction and in general performs better than steepest descent.

In practice, experiments have *multiple responses* so GRSM is more relevant than RSM. GRSM generalizes steepest descent (applied in RSM) through ideas from *interior-point* methods in mathematical programming. This search direction moves faster to the optimum than steepest descent, since the GRSM avoids creeping along the boundary of the feasible area determined by the constraints on the random outputs and the deterministic inputs. GRSM's search direction is scale independent. More specifically, this *search direction* is

$$\mathbf{d} = -\left(\mathbf{B}'\mathbf{S}^{-2}\mathbf{B} + \mathbf{R}^{-2} + \mathbf{V}^{-2}\right)^{-1}\widehat{\beta}_{0;-0}, \tag{3.10}$$

where $\mathbf{B}$ is the matrix with the gradients of the constrained outputs, $\mathbf{S}$, $\mathbf{R}$, and $\mathbf{V}$ are diagonal matrixes with the current estimated slack values for the constrained outputs, and the lower and upper limits for the deterministic inputs, and $\widehat{\beta}_{0;-0}$ is the classic estimated steepest descent direction for the goal variable with index 0; the remaining responses have indexes $1, \ldots, r-1$.

Analogously to RSM, GRSM proceeds *stepwise*; i.e., after each step along the search path (3.10), the following hypotheses are tested:

1. The observed goal output of the new combination is suboptimal; i.e. it gives *no improvement* compared with the old combination (pessimistic null hypothesis).
2. This new combination is *feasible*; i.e., the other random outputs satisfy the constraints (optimistic null hypothesis).

To test these hypotheses, the classic Student t statistic may be applied (a paired t statistic if CRN are used; see Law (2007, p. 553). Because multiple hypotheses are tested, Bonferroni's inequality may be used; i.e., divide the classic α value (e.g., $\alpha = 0.10$) by the number of tests.

Actually, a better combination may lie in between the old combination and the new combination. Therefore GRSM uses *binary search*; i.e., it observes a combination that lies halfway between these two combinations (and is still on the search path defined by (3.10)). This halving of the stepsize may be applied a number of times.

Next, GRSM proceeds analogously to RSM. So, around the best combination found so far, it selects a new local area. It uses a R-III design to select new input combinations to be observed. It fits first-order polynomials for each of the r outputs, which gives a *new* search direction (3.10). And so GRSM goes on.

In experiments with random outputs it is necessary to estimate the gradients and the slacks of the constraints; see (3.10). This estimation turns the KKT first-order optimality conditions (which are classic in nonlinear mathematical programming) into a problem of nonlinear statistics. Angün and Kleijnen (2009) derive an asymptotic test; Bettonvil et al. (2009) derive a small-sample bootstrap test. GRSM has been applied to an (s, S) inventory system and a toy problem consisting of second-order polynomials augmented with additive multivariate normal noise. Kleijnen and Wan (2007) compare GRSM with OptQuest—a popular commercial simulation-optimization heuristic that combines the metaheuristics of tabu search, neural networks, and scatter search; also see the references to these metaheuristics in Sect. 3.1.

3.5.2 Kriging and Mathematical Programming

Kleijnen et al. (2010) present a novel heuristic for constrained optimization in black-box experimentation (an alternative approach is GRSM, discussed in Sect. 3.5.1). Moreover, they assume that the inputs must be *integers*. Their heuristic combines (i) sequential designs to specify the inputs, (ii) Kriging metamodels to analyze the global I/O (whereas GRSM uses local metamodels), and (iii) integer nonlinear programming (INLP) to estimate the optimal solution from the Kriging metamodels. They call this heuristic DOE-Kri-MP and apply it to a toy problem and an (s, S) inventory system (similar to the the GRSM examples) and a realistic call-center simulation; they also compare this heuristic with OptQuest and find that their heuristic requires fewer input combinations.

As Fig. 3.1 shows, their heuristic starts with the selection of an initial—or pilot—space-filling design, and observes the corresponding outputs. Because the outputs are random, enough replicates are observed to realize a prespecified relative precision, following Law (2007, pp. 500–503). For each combination of this design, the heuristic obtains (say) r average outputs $\overline{w_h}$ $(h = 0, .., r - 1)$ where the subscript "0" denotes the goal output and the other subscripts the constrained outputs. Next, the heuristic fits r univariate Kriging metamodels to this I/O. These metamodels are validated; as long as one or more metamodels are proclaimed not to be valid, the current design is augmented, observing a new combination in the global search area, to fine-tune the metamodels. The heuristic then refits the Kriging metamodels using the augmented I/O. When all r Kriging metamodels are accepted, the heuristic applies an INLP program (namely, Matlab's free program called bnb20) to the r Kriging metamodels to estimate the optimum. The heuristic checks whether the estimated optimum has already been observed before; if it has, then the heuristic reruns the INLP to estimate a new next best point, i.e., all points already observed are excluded from the optimization. Anyhow, the new point is observed and its I/O data are added to the current design. Next, the heuristic stops if the output data of the new point compared with the output of the best point found so far shows that the last (say) 30 times INLP solutions do not give a combination with a significant improvement in the objective function. Otherwise, the Kriging metamodels are

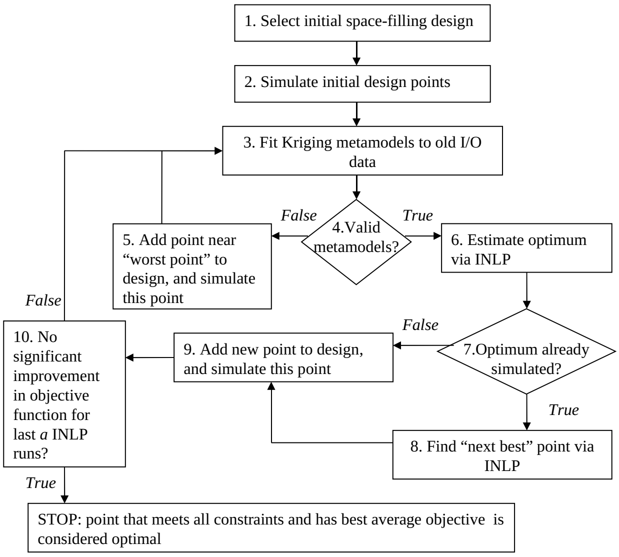

1. Select initial space-filling design
2. Simulate initial design points
3. Fit Kriging metamodels to old I/O data
False
4.Valid metamodels?
True
5. Add point near "worst point" to design, and simulate this point
6. Estimate optimum via INLP
False
False
10. No significant improvement in objective function for last *a* INLP runs?
9. Add new point to design, and simulate this point
7.Optimum already simulated?
True
8. Find "next best" point via INLP
True
STOP: point that meets all constraints and has best average objective is considered optimal

Fig. 3.1: Overview of DOE-Kri-MP

updated using old and new I/O data, and the heuristic continues its search. Some details are as follows:

1. The *pilot* design uses a type of LHS, which accounts for box constraints for the inputs. Moreover, the design is adapted to account for non-box input constraints; e.g., the sum of some inputs must meet a budget constraint.
2. To *validate* the Kriging metamodels, the heuristic applies *cross-validation*. To estimate the variance of the Kriging predictor, the heuristic applies distribution-free bootstrapping to the replicates (accounting for a variable number of replicates per input combination, and CRN).
3. Whereas some combinations are selected to improve the *global* Kriging metamodel (global search), some other combinations are added because they seem to be close to the *local optimum* (local search).

3.5.3 Taguchian Robust Optimization

Originally, *Taguchi* developed his approach to help Toyota design robust cars; i.e., cars that perform reasonably well in many environments (from the snows in Alaska to the sands in the Sahara); see Taguchi (1987) and Wu and Hamada (2000). He distinguishes between two types of variables:

- Decision (or control) factors (say) d_j $(j = 1, \dots, k)$.
- Environmental (or noise) factors, e_g $(g = 1, \dots, c)$.

Most optimization methods assume a known environment; i.e., the problem for which an optimal solution should be found is given and fixed. Robust optimization algorithms, however, account for the uncertainty of the environment; e.g., the optimization algorithm should be applicable to new, unknown problems. Related (but different) is robust mathematical programming: whereas classic mathematical programming assumes that all coefficients are known, robust mathematical programming assumes that the coefficients may vary over a specific area, and tries to find solutions that give nearly optimal output even when the coefficients vary within that area; see Ben-Tal and Nemirovski (2002).

We advocate a methodology that uses Taguchi's view of the uncertain world, but replaces his statistical techniques by either RSM or Kriging combined with mathematical programming. Myers and Montgomery (1995) extend RSM to robust optimization of real-life systems. Dellino et al. (2009) adapt this robust RSM to simulated systems; instead of RSM they also apply Kriging. They apply their methods to an *economic order quantity* (EOQ) inventory model, which is a classic model in operations research.

Taguchi uses a *scalar* output such as the signal-to-noise or mean-to-variance ratio, whereas Dellino et al. allow each output to have a statistical distribution characterized through its mean and standard deviation; also see Myers and Montgomery (1995, p. 491). Dellino et al. minimize the mean while the standard deviation remains below a given threshold value; they solve this constrained optimization problem using some nonlinear programming (NLP) code (namely Matlab's free `fmincon`). Next they change the threshold value for the standard deviation, which gives the estimated *Pareto frontier*.

Now we present some mathematical details. Inspired by RSM (see Sect. 3.5.1), Myers and Montgomery (1995, p. 218, 492) assume:

- a *second-order* polynomial for the decision factors d_j,
- a first-order polynomial for the environmental factors e_g,
- *control-by-noise two-factor interactions* (say) $\delta_{j;g}$,

resulting in the response surface (metamodel)

$$\begin{aligned} y = \beta_0 + \sum_{j=1}^{k}\beta_j d_j + \sum_{j=1}^{k}\sum_{j' \geq j}^{k}\beta_{j;j'} d_j d_{j'} + \\ + \sum_{g=1}^{c}\gamma_j e_j + \sum_{j=1}^{k}\sum_{g=1}^{c}\delta_{j;g} d_j e_g + \epsilon \\ = \beta_0 + \beta' \mathbf{d} + \mathbf{d}'\mathbf{B}\mathbf{d} + \gamma'\mathbf{e} + \mathbf{d}'\boldsymbol{\Delta}\mathbf{e} + \epsilon, \end{aligned} \tag{3.11}$$

where y denotes the regression predictor of the experiment's output w, ϵ the regression residual, $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_k)'$, $\mathbf{d} = (d_1, \ldots, d_k)'$, $\mathbf{B}$ the $k \times k$ symmetric matrix with main-diagonal elements $\beta_{j;j}$ and off-diagonal elements $\beta_{j;j'}/2$, $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_c)'$, $\mathbf{e} = (e_1, \ldots, e_c)'$, and $\boldsymbol{\Delta} = (\delta_{j;g})$.

Myers and Montgomery (1995, p. 493–494) assume that the environmental variables $\mathbf{e}$ have zero mean and constant variance; moreover they are not correlated: $E(\mathbf{e}) = \mathbf{0}$ and $\mathbf{cov}(\mathbf{e}) = \sigma_e^2\mathbf{I}$. Dellino et al. replace these assumptions by a more general assumption: $E(\mathbf{e}) = \mu_{\mathbf{e}}$ and $\mathbf{cov}(\mathbf{e}) = \boldsymbol{\Omega}_{\mathbf{e}}$. Next they derive from (3.11)

$$E(y) = \beta_0 + \beta'\mathbf{d} + \mathbf{d}'\mathbf{B}\mathbf{d} + \gamma'\mu_{\mathbf{e}} + \mathbf{d}'\boldsymbol{\Delta}\mu_{\mathbf{e}} \tag{3.12}$$

and

$$var(y) = (\gamma' + \mathbf{d}'\boldsymbol{\Delta})\boldsymbol{\Omega}_{\mathbf{e}}(\gamma + \boldsymbol{\Delta}'\mathbf{d}) + \sigma_\epsilon^2 = \mathbf{l}'\boldsymbol{\Omega}_{\mathbf{e}}\mathbf{l} + \sigma_\epsilon^2. \tag{3.13}$$

where $\mathbf{l} = (\gamma + \boldsymbol{\Delta}'\mathbf{d}) = (\partial y/\partial e_1, \ldots, \partial y/\partial e_c)'$; i.e., $\mathbf{l}$ is the gradient with respect to the environmental factors. So, the larger the gradient's components are, the larger the variance of the predicted output is. Furthermore, if $\boldsymbol{\Delta} = \mathbf{0}$ (no control-by-noise interactions), then $var(y)$ cannot be controlled through the control variables $\mathbf{d}$.

To estimate the parameters in (3.12) and (3.13), Dellino et al. use OLS; see (3.2). Myers and Montgomery (1995, pp. 463–534) use only two values per environmental factor, which suffices to estimate its main effect and its interactions with the decision factors in (3.11). Dellino et al., however, use LHS to select values for the environmental factors. These values are crossed with the values for the decision variables; such crossing is usual in Taguchian designs. Designs more efficient than crossed designs are discussed by Dellino et al. (2009) and Myers and Montgomery (1995, p. 487).

The goal of Dellino et al.'s robust optimization is to minimize the resulting estimated mean $\widehat{y}$, while keeping the estimated standard deviation $\widehat{\sigma_y}$ below a given threshold. This constrained minimization problem they solve through mathematical programming. This gives the values of the estimated robust decision variables (say) $\widehat{\mathbf{d}^+}$ and its corresponding estimated optimal mean $\widehat{y}$ and standard deviation $\widehat{\sigma_y}$. Next, they vary the threshold value for the standard deviation (say) 100 times, which may give up to 100 different solutions $\widehat{\mathbf{d}^+}$ with their corresponding $\widehat{y}$ and $\widehat{\sigma_y}$. These pairs $(\widehat{y}, \widehat{\sigma_y})$ give the estimated Pareto frontier. To estimate the variability of this frontier, they use *parametric bootstrapping*; i.e., they assume that the type of distribution of the relevant random variable is known (in their EOQ example, the distribution is Gaussian)—the parameters of that distribution are estimated from the I/O data resulting from the experiment. Note that Dellino et al. (2009) also use Kriging instead of RSM.

Future research may address the following issues:

- Instead of deterministic simulation experiments, we may investigate experiments with *random* output—either a scalar or a vector.
- *Integer* constraints on some inputs may apply so INLP may be used (as in Sect. 3.5.2).

3.6 Conclusions

In this chapter we have presented an overview of the design and analysis of computational experiments with optimization algorithms. We covered classic designs and their corresponding metamodels; namely, R-III designs (such as 2_{III}^{k-p} designs) for first-order polynomials, R-IV and R-V designs for two-factor interactions, and CCD for second-degree polynomials. These designs may be applied when the experimental area is relatively small, so a low-order polynomial is an adequate approximation of the I/O function in the experiment (Taylor series argument).

We also reviewed factor screening in experiments with very many factors, assuming that only a few factors are really important. We focused on the sequential bifurcation method, which assumes a first-order polynomial, possibly augmented with two-factor interactions. For didactic reasons we presented screening after classic designs and analyses; in practice, however, we recommend to start with a screening experiment rather than using intuition and experience to select a small set of factors.

Furthermore, we reviewed Kriging and space-filling designs (e.g., LHS). A Kriging model may give a better approximation when the experimental area is relatively large. Finally, we discussed experiments aimed at optimization, allowing for experiments that give multiple random outputs. This optimization may use Generalized RSM or Kriging combined with mathematical programming. We ended with Taguchian robust optimization.

So, which designs to use depends on the metamodel to be used. The metamodel choice depends on the goal of the experiment; e.g., optimization based on RSM uses a sequence of first-order polynomials that give the steepest descent search direction (so R-III designs suffice). If the goal is sensitivity analysis and the experimental area is considered to be large, then Kriging may be useful; Kriging may start with a space-filling design (e.g., LHS). We gave selected references for further study; Kleijnen (2008) lists more than 400 references and gives website addresses for software. We also listed topics for future research, in various sections.

What remains is the application of these various designs and analysis methods to computational experiments (applications to other types of experiments are summarized by Kleijnen (2008)). Some guidelines are given by Kleijnen et al. (2005), but there are no cookbook recipes: learning by doing is the only route to skillful experimentation.

Acknowledgements Comments by two anonymous referees led to a much improved version of the present chapter.

References

Adenso-Diaz B, Laguna M (2006) Fine-tuning of algorithms using fractional experimental designs and local search. Operations Research 54(1):99–114

Angün E, Kleijnen J (2009) An asymptotic test of optimality conditions in multiresponse simulation-based optimization, working paper

Angün E, Kleijnen J, den Hertog D, Gürkan G (2009) Response surface methodology with stochastic constrains for expensive simulation. Journal of the Operational Research Society 60:735–746

Ankenman B, Nelson B, Staum J (2009) Stochastic kriging for simulation metamodeling. Operations Research (accepted)

Bartz-Beielstein T (2006) Experimental research in evolutionary computation—The new experimentalism. Natural Computing Series, Springer

Bartz-Beielstein T, Preuss M (2010) The future of experimental research. In: Bartz-Beielstein T, Chiarandini M, Paquete L, Preuss M (eds) Empirical methods for the analysis of optimization algorithms, Springer, pp 17–46

Ben-Tal A, Nemirovski A (2002) Robust optimization: methodology and applications. Mathematical Programming 92(3):353–380

Bettonvil B, Kleijnen J (1996) Searching for important factors in simulation models with many factors: sequential bifurcation. European Journal of Operational Research 96:180–194

Bettonvil B, del Castillo E, Kleijnen J (2009) Statistical testing of optimality conditions in multiresponse simulation-based optimization. European Journal of Operational Research 199(2):448–458

Box G, Wilson K (1951) On the experimental attainment of optimum conditions. Journal Royal Statistical Society, Series B 13(1):1–38

Cressie N (1993) Statistics for spatial data: revised edition. Wiley, New York

Dellino G, Kleijnen J, Meloni C (2009) Robust optimization in simulation: Taguchi and response surface methodology. In: Rossini M, Hill R, Johansson B, Dunkin A, Ingalls R (eds) Proceedings of the 2009 Winter Simulation Conference, (accepted)

Den Hertog D, Kleijnen J, Siem A (2006) The correct Kriging variance estimated by bootstrapping. Journal of the Operational Research Society 57(4):400–409

Efron B, Tibshirani R (1993) An introduction to the bootstrap. Chapman & Hall, NY

Fu M (2008) What you should know about simulation and derivatives. Naval Research Logistics 55:723–736

Kleijnen J (2008) Design and analysis of simulation experiments. Springer

Kleijnen J, Sargent R (2000) A methodology for the fitting and validation of metamodels in simulation. European Journal of Operational Research 120(1):14–29

Kleijnen J, Van Beers W (2009) Monotonicity-preserving bootstrapped Kriging metamodels for expensive simulations, working paper

Kleijnen J, Wan J (2007) Optimization of simulated systems: OptQuest and alternatives. Simulation Modelling Practice and Theory 15:354–362

Kleijnen J, Sanchez S, Lucas T, Cioppa T (2005) State-of-the-art review: a user's guide to the brave new world of designing simulation experiments. INFORMS Journal on Computing 17(3):263–289

Kleijnen J, Bettonvil B, Persson F (2006a) Screening for the important factors in large discrete-event simulation: sequential bifurcation and its applications. In: Dean A, Lewis S (eds) Screening: Methods for experimentation in industry, drug discovery, and genetics, Springer, pp 287–307

Kleijnen J, den Hertog D, Angün E (2006b) Response surface methodology's steepest ascent and step size revisited: correction. European Journal of Operational Research 170:664–666

Kleijnen J, Van Beers W, Van Nieuwenhuyse I (2010) Constrained optimization in simulation: a novel approach. European Journal of Operational Research 202:164–174

Law A (2007) Simulation modeling and analysis, 4th edn. McGraw-Hill, Boston

Lophaven S, Nielsen H, Søndergaard J (2002) DACE—A Matlab Kriging Toolbox. Tech. Rep. IMM-REP-2002-12, Informatics and Mathematical Modelling, Technical University of Denmark, Copenhagen, Denmark

Montgomery D (2009) Design and analysis of experiments, 7th edn. Wiley, Hoboken, New Jersey

Myers R, Montgomery D (1995) Response surface methodology: process and product optimization using designed experiments. Wiley, NY

Park S, Fowler J, Mackulak G, Keats J, Carlyle W (2002) D-optimal sequential experiments for generating a simulation-based cycle time-throughput curve. Operations Research 50(6):981–990

Rajagopalan HK, Vergara FE, Saydam C, Xiao J (2007) Developing effective meta-heuristics for a probabilistic location model via experimental design. European Journal of Operational Research 177(1):83–101, URL `http://dx.doi.org/10.1016/j.ejor.2005.11.007`

Ridge E, Kudenko D (2007) Screening the parameters affecting heuristic performance. In: Lipson H (ed) GECCO, ACM, p 180, URL `http://doi.acm.org/10.1145/1276958.1276994`

Ridge E, Kudenko D (2010) Sequential experiment designs for screening and tuning parameters of stochastic heuristics. In: Bartz-Beielstein T, Chiarandini M, Paquete L, Preuss M (eds) Empirical Methods for the Analysis of Optimization Algorithms, Springer, pp 265–287

Sacks J, Welch WJ, Mitchell TJ, Wynn HP (1989) Design and analysis of computer experiments. Statistical Science 4(4):409–435

Santner TJ, Williams BJ, Notz WI (2003) The design and analysis of computer experiments. Springer

Taguchi G (1987) System of experimental designs. Krauss International, NY

Van Beers W, Kleijnen J (2003) Kriging for interpolation in random simulation. Journal of the Operational Research Society (54):255–262

Van Beers W, Kleijnen J (2008) Customized sequential designs for random simulation experiments: Kriging metamodeling and bootstrapping. European Journal of Operational Research 186(3):1099–1113

Wu C, Hamada M (2000) Experiments; planning, analysis, and parameter design optimization. Wiley, NY

Xu J, Yang F, Wan H (2007) Controlled sequential bifurcation for software reliability study. In: Henderson S, Biller B, Hsieh MH, Shortle J, Tew J, Barton R (eds) Proceedings of the 2007 Winter Simulation Conference, pp 281–288

Yin J, Ng S, Ng K (2008) Kriging model with modified nugget effect. In: Proceedings of the 2008 IEEE International Conference on Industrial Engineering and Engineering Management, pp 1714–1718

Yu HF (2007) Designing a screening experiment with a reciprocal Weibull degradation rate. Computers & Industrial Engineering 52(2):175–191

Chapter 4
The Generation of Experimental Data for Computational Testing in Optimization

Nicholas G. Hall and Marc E. Posner

Abstract This chapter discusses approaches to generating synthetic data for use in scientific experiments. In many diverse scientific fields, the lack of availability, high cost or inconvenience of the collection of real-world data motivates the generation of synthetic data. In many experiments, the method chosen to generate synthetic data can significantly affect the results of an experiment. Unfortunately, the scientific literature does not contain general protocols for how synthetic data should be generated. The purpose of this chapter is to rectify that deficiency. The protocol we propose is based on several generation principles. These principles motivate and organize the data generation process. The principles are operationalized by generation properties. Then, together with information about the features of the application and of the experiment, the properties are used to construct a data generation scheme. Finally, we suggest procedures for validating the synthetic data generated. The usefulness of our protocol is illustrated by a discussion of numerous applications of data generation from the optimization literature. This discussion identifies examples of both good and bad data generation practice as it relates to our protocol.

4.1 Introduction

Most scientific models require the use of data for evaluation and validation. The three major sources that researchers use to obtain this data are the real world, library databanks, and random generation procedures. A natural choice is to use real-world data. One advantage of using this type of data is its relevance, which enables esti-

Nicholas G. Hall
Department of Management Sciences, The Ohio State University, Columbus, Ohio, USA
e-mail: hall_33@fisher.osu.edu

Marc E. Posner
Department of Integrated Systems Engineering, The Ohio State University, Columbus, Ohio, USA
e-mail: posner.1@osu.edu

mation of the eventual practical usefulness of the results of an experiment. A second advantage of using real-world data is that it enhances the credibility of the experiment. Moreover, this type of data is usually unbiased. However, there can be serious difficulties with using real-world data. First, it may not be available, or not available in quantities sufficient for the purposes of the experiment. This situation typically arises when a research area is new, or when the phenomenon being examined is fragile. An example of the latter situation is data collected from destructive testing. Further, most real-world data for a given application are generated by the same physical process. This may limit the size and variety of the data sets. For example, the amount of inventory data for a given product typically does not exceed the number of working days in a year. Moreover, each point is generated by the same underlying production and demand processes. There are also cases where very large and varied data sets are needed for testing (Wei et al. 2003). Further, there are situations where the collection of real-world data is costly, slow or inconvenient. Another concern is that, even when real-world data is available, it may be preferable to use data that is free from the stochastic vagaries of the real environment (Kadlec 2000). A further issue with real world data is that it can impose a very special, restricted structure on the problem (Fischetti et al. 2001).

There are many applications for which library data is available. Examples occur in the testing of computer systems and algorithms. Among the advantages of library data sets are that some properties of the problem, such as an optimal solution and the performance of other procedures on the data set, are known. Also, there is a recent literature on determining which data from an existing data set should be sampled to meet the objectives of a particular experiment (Koehler and Owen 1996). However, there are many applications where an adequate data library does not exist (McGeoch 1996). Another concern is that library data sets are sometimes biased by the inclusion of problems for which a particular previous experiment gives good results (Hooker 1995). A final problem is that "you get what you measure." A comparison of procedures based on a set of library data leads to selection of procedures that work efficiently for that set. The selected procedures may work poorly for more typical types of data.

When it is not possible or desirable to use real-world or library data, an alternative is to generate *random* data. This type of data is also known as *synthetic* data. Generally, synthetic data is generated by a computer program using a pseudorandom number generator. Sometimes synthetic data is generated by perturbing real-world or library data. This protects confidentiality (Reiter 2002), introduces uncertainty into a deterministic problem (Bertsimas et al. 2006), and generates additional test data (Bienstock et al. 2006, Linderoth et al. 2006, Pan and Shi 2007). Some authors view synthetic data generation as a last resort (Degraeve and Schrage 1997). However, the substantial advantages of synthetic data include the speed and ease of generation, and the quantity, variety, and relevance of data that can be obtained. As an example of the latter, extreme problem instances that are needed to study robust performance can be generated efficiently using synthetic data (Yaman et al. 2007).

Applications of synthetic data generation can be found in many diverse scientific fields. Examples include: acoustics (Hooshyar et al. 2000), astrophysics (McIntosh et al. 2000), analytical chemistry (Grate et al. 1999), cluster analysis (Krieger and Green 1999), computer imaging (Wilson and Hancock 2000), crystallography (Roversi et al. 1998), database design (Beyer et al. 1999), data mining (Shen et al. 1999), environmental engineering (Hooker 1995), evolutionary trees (Pearson et al. 1999), fuzzy systems (Ray and Ghoshal 2000), hydrology (Aksoy and Bayazit 2000), marketing (Bijmolt and Wedel 1999), medicine (Freed 2000), meteorology (Yuval 2000), mineral processing (Schena and Chiaruttini 2000), neural networks (Bauer et al. 1999), pattern recognition (Ho and Baird 1997), physics (Ruchala et al. 1999), radio science (Herique 1999), robotics (Gonzalez and Gutierrez 1999), seismology (Lu et al. 1997), statistical inference (Qin and Jing 2000), time series (Goutte 2000) and transportation planning (Munizaga et al. 2000). The frequency with which synthetic data is generated, the impressive variety of applications, and the considerable impact of the data generation process on experimental results all emphasize the importance of proper data generation methods. In spite of this, there are no general protocols for the generation of synthetic data.

Because of the lack of general guidelines, the scientific literature contains many examples where difficulties arise in synthetic data generation. For instance, in cluster analysis, standard random sampling techniques frequently miss small clusters (Palmer and Faloutsos 2000). Also, there is evidence that synthetic surface seismic data show a poor qualitative match with real data (Frenje and Juhlin 1998). As another example, most noise components generated in synthetic data are similar to Gaussian white noise, whereas the noise that occurs in real data typically follows a more complex and varied pattern (Gelius and Westerdahl 1997). Finally, as discussed in Hall and Posner (2001), there are experiments that test computer algorithms where the synthetic data contains instances for which the optimal values of most variables can be found trivially (Potts and Van Wassenhove 1988). Additional examples of difficulties with data generation schemes are discussed in Sect. 4.3.

A frequent goal of synthetic data generation is to generate a "typical" class of problems. However, for most optimization applications, such a class is hard to characterize. For many experiments, this goal may even be inappropriate. A better approach is to generate classes of problems with different characteristics and investigate how the performance of a procedure depends on these characteristics (Hooker 1994, Hall and Posner 2007). Moreover, the generation of synthetic data sets in a controlled way represents an important step towards a more scientific approach to testing optimization procedures. This approach to data generation is supported by the literature on "no free lunch theorems" (Schaffer 1994, Wolpert and Macready 1997). Such theorems show that, if a procedure achieves superior results on some types of instances, it must compensate with inferior performance on other types of instances. This suggests that matching procedures to types of instance results in better performance than applying a common method across all instances. This idea is demonstrated by Hall and Posner (2007), Smith-Miles (2008), Smith-Miles et al. (2009). It further suggests that an arbitrary synthetic data generation procedure provides little useful information about the relative performance of optimization procedures.

Synthetic data is needed for a variety of purposes that include hypothesis testing, evaluation, design improvement, optimization, and software testing. In *hypothesis testing*, we may wish to investigate which of several alternative hypotheses best explains a natural phenomenon, which of several competing drugs is most effective at fighting a specific disease, or which computer algorithm runs fastest. A second purpose is *evaluation*. For example, we may wish to observe the cooling pattern of a metal in an injection mold, the spread of environmental pollution, or the behavior of a meteorological phenomenon. In these situations, synthetic data may be needed to perform a simulation study. Another purpose is *design improvement*. For example, we may test components or prototype versions of a system, product or algorithm at intermediate design stages. The objective is to gain information that improves the final design. Such testing is often classified as a pilot study. For example, we may wish to test an algorithm to ensure that it is robust for a variety of types of data. Random data is often needed because the uniqueness of the design process results in a lack of real-world data. Examples of systems and products where this may occur include designing a manufacturing system, increasing the sensitivity of a medical diagnostic procedure, and improving the performance of a laser weld. Synthetic data is also useful for solving *optimization* problems. For example, one standard approach for solving stochastic programming problems is to generate scenario trees using synthetic data. These trees discretize the continuous space of probabilistic problem scenarios. The scenario trees define stochastic programming problem instances that are solved in order to provide an approximate optimal solution. *Software testing*, which is closely related to optimization, is another purpose of synthetic data. This type of testing ensures the correctness of a software program. For this application, synthetic data must be generated so that the correctness of each line of software code is tested.

This chapter focuses on the generation of synthetic data to test the performance of optimization heuristics and algorithms. The scope includes hypothesis testing, evaluation, design improvement, and optimization. While testing software programs that implement optimization procedures is of practical importance, it imposes very different requirements on the data that is generated. As a result, we do not consider this type of synthetic data generation.

The chapter is organized as follows. In Sect. 4.2, we propose a general protocol for synthetic data generation. Sect. 4.3 discusses numerous applications of synthetic data generation for the testing of optimization algorithms. These applications illustrate both good and bad practices of synthetic data generation as it relates to our protocol. Sect. 4.4 provides a conclusion and some suggestions for future research.

4.2 A Protocol

A *data generation scheme* is a complete step-by-step specification of the procedure by which data is generated, including the parameter ranges and the random number generation method. Our protocol for synthetic data generation is a multilevel structure that consists of *generation principles*, *features of the application*, *features of the experiment*, *generation scheme*, and *validation*. The generation principles provide high-level perspectives on the data generation process and are independent of the details of a particular application or experiment. To provide more detailed guidelines that ensure the principles are satisfied, we develop *generation properties* to operationalize each generation principle. With the properties as guidelines for good practice, the features of the application and the features of the experiment determine the actual generation scheme. After the data is generated, we examine the *attributes* of the data to determine whether the data is suitable. To accomplish this, we must be able to quantify and measure these attributes. An overview of the protocol we propose appears in Fig. 4.1. The structure that we develop is used to discuss and evaluate generation schemes in Sect. 4.3.

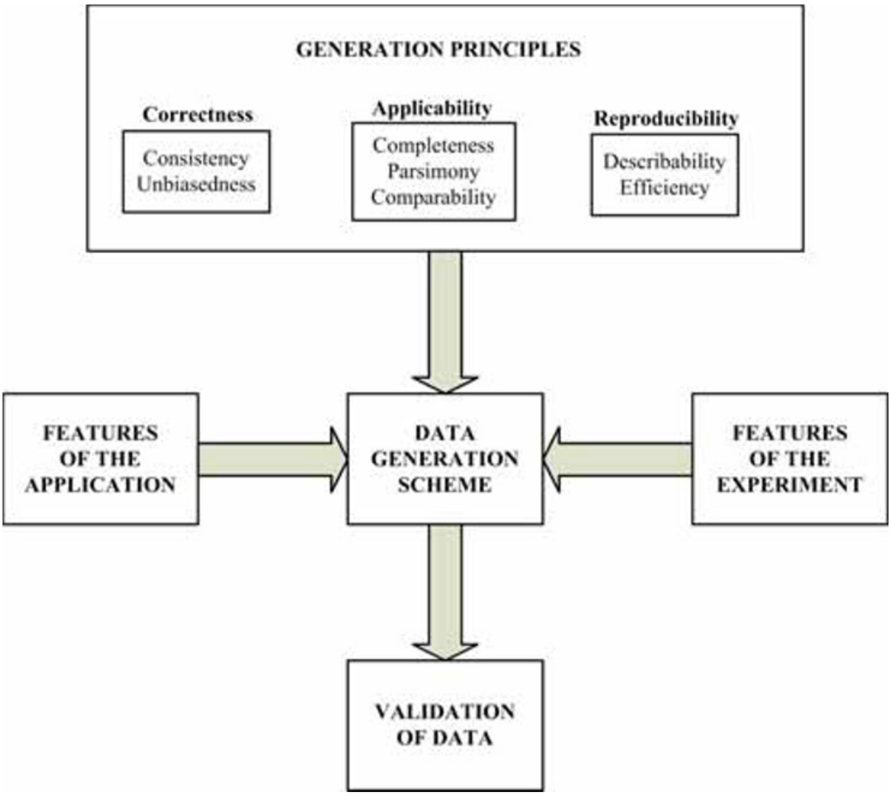


Fig. 4.1: A protocol for synthetic data generation

4.2.1 Generation Principles and Properties

Generation principles are high-level policies that should be followed when generating synthetic data. To operationalize these principles, we provide *generation properties*. These are more specific requirements that can be incorporated into any data generation scheme.

Three principles that should be present in a data generation scheme are:

1. Correctness: The data sets generated are free from defects.
2. Applicability: The generation scheme generates the types of data sets that are needed.
3. Reproducibility: The generation scheme and its data sets are reproducible.

It is important that the data is constructed in a way that is logical and free of distortions. Otherwise, the results of the analysis are not valid. Two generation properties that operationalize the principle of Correctness are:

i. Consistency: Identical types of data are generated in the same way.
ii. Unbiasedness: All biases in the data are controlled.

The Consistency property requires that all elements of the same type of data are generated in a similar way. Thus, the generation scheme should not provide different treatment for particular elements of the data. For example, assume that the mean of a particular element has a desired value. Consider a scheme that randomly generates data for all but the last element and then sets the value of the last element to achieve the desired mean. This procedure treats the last element differently from the others. Consequently, it is inconsistent and may have unpredictable and biased effects on the results of an experiment. A more consistent way to generate the values of the elements is to set an expected mean for the elements and allow some variance around that value (Hall and Posner 2001).

The Unbiasedness property requires that the data is generated in a way that does not promote some experimental results over others. In testing the relative performance of two or more algorithms, for example, certain characteristics of the data may favor a particular algorithm. Similarly, when evaluating a heuristic, the use of biased data may lead to erroneous conclusions. To formalize Unbiasedness, it is helpful to regard all possible data sets that can be generated by a particular synthetic data generation scheme as members of a population. The generation of a particular data set is analogous to the selection of a sample from this population. Then, Unbiasedness can be seen as a requirement that the sample is a fair representation of the scenarios being modeled. When constructing synthetic data, it is important not to introduce inadvertent biases into the generation scheme. However, this is not always possible. For example, as mentioned by Hadjar et al. (2006), the special structure of some real-world vehicle routing problems is hard to reproduce synthetically. Hence, the generated data may be biased.

The second generation principle is Applicability. The purpose of the experiment, such as hypothesis testing, evaluation, design improvement or optimization, may greatly affect the way in which the data is generated. The Applicability criterion requires that the data is generated to satisfy this purpose. For example, suppose an experiment involves testing an algorithm over specific data sets that are likely to occur in practice. Then, the data generation scheme should produce data sets that are representative of these particular situations. Another example is a comparison between two algorithms. This experiment needs a varied set of data that represents all possible data sets to which these algorithms could be applied. As another example, complex interactions between components in the design of a heuristic procedure may generate issues that a designer needs to address. The generation of customized synthetic data that models these interactions may aid in the resolution of those issues. A final example occurs in generating a scenario tree for stochastic programming. To find solutions that are close to optimal, the scenario tree should discretize the solution space to a reasonable degree of accuracy, but be of a tractable size.

Three generation properties that operationalize the principle of Applicability are:

i. Completeness: All data sets that are important to the experiment can be generated.
ii. Parsimony: The variations in the data sets are important to the experiment.
iii. Comparability: The experiments are comparable within and between studies.

It is important that a generation scheme has the property of Completeness. A good generation scheme can generate all data sets that are relevant to the specific scientific experiment. The amount of variety needed in the data is normally determined by the objectives of the experiment. If the objective is to obtain good results for specific types of data, then little variety is necessary. However, if the objective is to design an algorithm that is robust, i.e., is expected to perform well for a wide range of scenarios, then much greater variety is necessary. Also, a generation scheme that fails to vary important causative variables appropriately may lead to the statistical phenomenon called autocorrelation (Hays 1973). In this case, the real relationships between the variables, as well as the significance of randomness in those relationships, may become impossible to estimate. To test the specific behavior of a procedure, different types of data sets are sometimes needed. In this case, the generation scheme should be sufficiently robust to generate all of the various types. However, if different generation schemes are used, then there may be issues with Comparability, as discussed below. If a goal is to use a procedure in a specific real-world setting, then the synthetic data should be similar to the real-world data. Therefore, the generation process should, if possible, emulate the way in which data is generated in the relevant real-world application.

Parsimony is a property that eliminates spurious variation within synthetic data. In general, the variety that occurs in a synthetic data set arises from only two sources: systematic variation of parameters as part of the experimental design, and randomness. Parameter variation that can have no effect on the experiment may distort the perceived effects of experimental variation. This can falsify the results. The Parsimony property is frequently important when a hypothesis is being tested through parametric changes to causative variables. An example is the testing of a branch-and-bound algorithm for integer programming. Doubling the size of all parameters in the problem leaves the problem unchanged. A generation procedure that considers parameter ranges which grow proportionately for all parameters simultaneously creates additional sets with the same type of data.

It is frequently important to be able to compare experiments. A lack of Comparability within a study may lead to erroneous or questionable conclusions. For example, due to problems in converting real-world data into homogeneous synthetic data, Lu et al. (1997) uses different procedures to construct data for small and large earthquakes. While this does not create difficulties when analyzing either the small or the large earthquakes separately, comparative statements about the two types of earthquakes become more problematic. Achieving Comparability with earlier studies is also important for the soundness of the experimental conclusions. An earlier study may be based on real-world data that is either no longer available, is already studied exhaustively, or is too small for current purposes. In this case, we may want to emulate the generation process of the earlier real-world data. Particular care is also needed when testing alternative hypotheses. A conclusion that one algorithm is preferred over another for a particular set of data should not be extrapolated to conclude that this preference holds over all possible data sets (Schaffer 1994, Wolpert and Macready 1997).

The principle of Reproducibility has long been recognized as important for scientific verification. When researchers want to validate results by reproducing experiments, they must be able to generate the same types of data that were originally generated. Usually, synthetic data is not described by a full specification of the data sets generated. Instead, a description of the underlying generation scheme is provided that is sufficiently detailed to permit reproduction. The conciseness of such descriptions is a valuable advantage of synthetic data (Hall and Posner 2001). Of course, the same synthetic data generation scheme can randomly generate sets of test problems with very different characteristics, particularly if sample sizes are small. Therefore, Reproducibility requires that the pseudorandom number generation scheme and random number seeds be part of the documentation.

Two generation properties that operationalize the principle of Reproducibility are:

i. Describability: The generation scheme is easy to describe.
ii. Efficiency: The generation scheme is easy and efficient to implement, use, and replicate.

For a data generation scheme to be describable, the description of the data generation scheme should be complete and comprehensible. If it is not, then researchers are unable to replicate the study. This property is typically more important when the experiment is not unique and when the results will be published. This is because uniqueness or confidentiality implies that it is less likely that there will be subsequent comparisons or evaluations of the experiment.

Efficiency can be measured either quantitatively or qualitatively and tends to have many possible levels of success. The goal of Efficiency is to develop a generation scheme that is as easy as possible to implement, use, and replicate. This is particularly important when large data sets are needed. For large data sets, a useful characteristic of synthetic data generation is the ability to link the data generation and experimentation stages so that it is necessary to store only one data set at a time. Ease of replication is also important for experiments that require large numbers of trials. For example, consider an experiment where randomly generated data sets from the same scheme have widely differing results. Then, many data sets may be needed to ensure the statistical significance of the conclusions. To validate these conclusions, statistical methods that determine the necessary sample size may be needed.

4.2.2 Features of the Model

Two important factors that determine which properties are important in a particular generation scheme are the *features of the application* and the *features of the experiment*. The features of the application are naturally occurring characteristics of the problem. These include the size of the data requirements, functional relationships between different parts of the data, and any bounds on the data. The features of the experiment are selected by the experimenter. These include the number and variety of tests, as well as intentions about implementation and publication. The features of the experiment may interact with the generation properties in various ways. For example, if only a single design is being evaluated, then Unbiasedness needs to be interpreted more narrowly. As another example, an algorithm may be designed for use in an application that is newly emerging and not yet well specified. For this application, the property of Completeness has less value, because it is unclear which data sets might be important to the experiment. As a final example, if the study is intended for commercial use and is not to be published, then Describability is less important.

Consequently, each generation property is relevant to a greater or lesser degree for a given experiment. As a result, we use the features of the model and experiment to select and modify the properties in order to guide the creation of the synthetic data generation scheme.

4.2.3 Validating the Data

A critical step in the data generation process is the validation of the synthetic data. One approach is to run a small preliminary study and check the reasonableness of the results. This approach is common when Monte Carlo simulation is used for experimental evaluation (Law and Kelton 1991). The main problem with this approach is that assessing reasonableness requires a prejudgment about the results of the experiment. An alternative is to compare the synthetic data against real-world data. However, as discussed above, a frequent reason for generating synthetic data is that no real-world data is available. Also, if only a small amount of real-world data is available, then the results may not be statistically reliable.

Consequently, a better approach is to validate the synthetic data outside the structure of the experiment. To do so requires identifying the key *attributes* of the data that may influence the results of the experiment. For example, among the relevant attributes of homogeneous numerical data are its average value and its dispersion. These attributes need to be evaluated and controlled in order to conduct a meaningful and efficient experiment. The existence of particular attributes in the synthetic data is often important. Where relevant, the values of these attributes should be similar to those in the real-world data, and should also match the experimental design. Some attributes of the data may be hard to specify and measure. An example is where the relationship between the attributes and the experimental results is unclear or complex (for example, highly nonlinear). Another example is where the attributes are not easily quantifiable, for instance the degree of clustering.

An attribute may be evaluated for one or more of the following reasons:

1. To control a causative variable in order to perform a hypothesis test
2. To ensure that the data varies so as not to bias an experiment
3. To document data variation that occurs naturally in practice
4. As a proxy for another attribute that is harder to define or measure

Each of these reasons implies the need for a *measure* to evaluate the attribute. For example, the measures of mean and variance evaluate the attributes of average value and dispersion, respectively. These measures can be computed using standard statistical methods. In experiments that study a cause-and-effect relationship between one or more causative variables and a dependent variable, measures are needed to control the values of the causative variables. Furthermore, a measure is needed to ensure that a level of variety consistent with the objectives of the experiment is introduced into the synthetic data. Also, when there is an attempt to reproduce real-world variation in the data, a measure of that variation is needed.

Sometimes, the direct measurement of an attribute is computationally difficult. For example, computing the number of feasible solutions in an intractable optimization problem may itself be an intractable problem (Garey and Johnson 1979). Consequently, rather than measure the exact amount of computation required by an enumerative algorithm for the 0-1 knapsack problem, Hall and Posner (2007) develop a heuristic procedure for estimating the number of nodes in the search tree.

Currently, there is no literature on the identification of suitable measures of the attributes. Therefore, an *ad hoc* approach is usually necessary. Some characteristics that a measure should have include:

1. Relevance to the experiment
2. Change in an appropriate way
3. Tractability

First, the measure should quantify the attributes of a data set in a way that is relevant to the outcome of the experiment. For example, in the testing of production planning algorithms, a simple heuristic performs best on many kinds of synthetic data. However, there are specific classes of data for which this is not true (Uma and Wein 1998). Thus, it would be valuable to determine and measure the important attributes of the data that could predict the performance of such a heuristic on a given data set. Second, the measure should not change spuriously with respect to the problem. For example, changing all costs proportionately in experimental testing of optimization software may leave the relative costs of all solutions unchanged. Consequently, the experimental results are unchanged also, and the measure should reflect this. Finally, since the purpose of a measure is to provide information about a data set, the evaluation of a measure should generally take significantly less time than it takes to perform the experiment itself.

As a result, for each important attribute, the data validation process should define one or more measures that can be used to determine to what extent the data contains that attribute. Comparisons between the synthetic data and any available real-world data are also valuable. For example, one relevant method is to test the hypothesis that the mean values of some attribute in the real data and the synthetic data are equal (Hays 1973). Finally, some experiments require several measures and these may be of different types. For example, if both computation time and accuracy are important to an experiment that tests the performance of a heuristic procedure then both attributes should be measured.

4.3 Applications to Optimization

In this section, we consider data generation issues that arise in various optimization applications. We identify several examples of good and bad data generation practice relative to the properties in our general protocol described in Sect. 4.2. Many of the data generation schemes we discuss are carefully designed, satisfy all of our criteria, and serve as excellent examples for future researchers to follow.

Unfortunately, in most synthetic data generation schemes, the generation properties are not all easy to satisfy. The Comparability property is usually easy to satisfy, except when both real-world and synthetic data are being used. Also, Describability is easy to satisfy in generation schemes for most optimization applications. An exception occurs in stochastic programming (see Sect. 4.3.7), where the underlying optimization problem formulations at each stage may need to be shown explicitly. Further, Efficiency is easy to satisfy for most optimization applications, except where large volumes of data are required. Stochastic programming is again an exception, due to the need to generate large scenario trees. Consequently, the four properties that require the most discussion in this section are Consistency, Unbiasedness, Completeness, and Parsimony.

A given generation property is easier to satisfy in some applications than in others. For example, data generation is easier when the required data has a simple structure that does not contain linking constraints or correlations. A second example occurs where real-world data has limited variation. This may imply that limited variation is also required in the synthetic data. A third example occurs where previous experiments have already documented a well-designed data generation scheme that satisfies all of the properties. However, trade-offs may arise between two or more properties. For example, a data generation scheme that is designed for Efficiency may have difficulty satisfying the Unbiasedness property (Sherali and Smith 2006). As another example, the random network generator in Demeulemeester et al. (2003) represents a trade-off between generating the data sets efficiently and generating all possible graphs with equal probability. The latter is required for Completeness. Failure to achieve Completeness may introduce biases into the data. In such situations, the features of the experiment should be used to determine a suitable compromise between the generation properties.

In view of the comments in the previous paragraph, we recognize that, in some applications, designing a data generation scheme that satisfies all of the properties may be difficult or even impossible. Consequently, our critical comments that appear with respect to some of the data generation schemes discussed below should be interpreted as comparisons with an *ideal* data generation scheme that may be very hard to develop. Moreover, failure to satisfy one or more of the properties we propose does not, by itself, suggest that the results of a research study are invalid. Rather, it suggests that additional testing is needed to determine whether the shortcomings of the data generation scheme have influenced the results.

Sections 4.3.1–4.3.7 are organized by application types. However, Sect. 4.3.8 contains a discussion of data generation issues that arise from the intractability of optimization problems across various application types.

4.3.1 Generalized Assignment and Knapsack

Solution procedures for the generalized assignment problem are proposed and tested computationally by Ross and Soland (1975), Chalmet and Gelders (1976), Ross and Soland (1977), Martello and Toth (1981), Trick (1992), and Racer and Amini (1994). A stochastic model for this problem is developed by Romeijn and Morales (2001a). This model analyzes the data generation schemes proposed by the earlier authors. The analysis reveals that, in all of these prior works, the generation schemes produce data sets where the capacity constraints become less tight as the number of machines increases. Such behavior makes the resulting problem instances easier to solve. Consequently, using the generation schemes used by Ross and Soland (1975, 1977), it is possible to conclude that a proposed greedy heuristic almost always finds the optimal solution. However, the use of a data generation scheme that satisfies Consistency shows that increases in the number of machines result in small increases in the relative error of the heuristic (Romeijn and Morales 2001a).

The issue of correlation between subsets of the data requires discussion. For the 0-1 linear knapsack problem (Balas and Zemel 1980, Martello and Toth 1988, 1997), and some one-machine scheduling problems (Potts and Van Wassenhove 1988, 1992), the authors report significantly longer computation times when the objective and constraint coefficients are positively correlated. Also, Guignard and

Rosenwein (1989), Trick (1992), and Amini and Racer (1994) report the same phenomenon when the objective function coefficients and the capacity constraint coefficients in the generalized assignment problem are negatively correlated. However, several of these research studies use data generation schemes that do not control the amount of correlation between the parameters. For example, the generation scheme used by Martello and Toth (1979) for the knapsack problem considers only data that is uncorrelated and data that has a correlation coefficient above $+0.97$. For the generalized assignment problem, the study of Martello and Toth (1981) considers only correlation coefficients of 0 and below -0.97. An issue with this type of data is that there is no information about how problem difficulty varies with correlation. An important question, which cannot be answered using the experimental designs in Martello and Toth (1979, 1981), is whether problem difficulty increases linearly with correlation or whether the increase occurs more quickly around some specific values of correlation. Other research studies that fail to control the correlation coefficient explicitly include those by Balas and Zemel (1980), Balas and Martin (1980), Potts and Van Wassenhove (1988), Guignard and Rosenwein (1989), John (1989), Rushmeier and Nemhauser (1993), and Amini and Racer (1994). Since the correlation coefficient strongly affects algorithmic performance, the computational studies in these works fail to satisfy the Completeness property. A procedure for approximately inducing prespecified Spearman rank correlation levels is proposed by Iman and Conover (1982). Also, a procedure for generating data with a prespecified Pearson correlation level between different subsets of data is developed by Hill and Reilly (1994). Both of these procedures are tested computationally on two-dimensional knapsack problems by Hill and Reilly (2000). They also find that correlation significantly affects the performance of both an optimal algorithm and a heuristic solution procedure. Because of the complexity of the procedure required to induce prespecified correlation levels, the Describability property is not easy to meet (Cario et al. 2002).

A study of a robust formulation of the 0-1 linear knapsack problem is considered by Atamtürk (2007). This study exclusively uses a knapsack size equal to half the item sizes. However, a study of the factors that affect computational difficulty in this problem shows that the knapsack capacity is one of the most important factors (Hall and Posner 2007). For this reason, it seems unlikely that the Completeness property can be satisfied without permitting the knapsack capacity to vary.

4.3.2 Supply Chains

A multiperiod single sourcing problem is considered by Romeijn and Morales (2001b). The authors derive a necessary and sufficient condition on the excess capacity such that the problem is feasible with probability one when the number of customers goes to infinity. This condition permits precise control of the tightness of the constraints in experimental data sets through the excess capacity parameter. Controlling the tightness of constraints as part of a generation scheme helps to satisfy the Completeness property by generating a wide variety of data sets. It also helps to satisfy the Unbiasedness property because uniform tightness of constraints may favor particular types of procedures. Similar comments can be made about a combined sourcing and scheduling problem (Chen and Pundoor 2006).

A subgradient search algorithm for multi-item, multifacility supply chain planning is tested computationally by Wu and Golbasi (2004). The large variety of combinations of parameters tested clearly satisfies the Completeness property. The following procedure is used to generate random capacity values. In the first period, the total demand for the period is multiplied by a constant larger than one to generate a value for capacity. In later time periods, total capacity assigned for the previous time periods is subtracted from total cumulative demand, and the result is then multiplied by a constant to generate capacity. This procedure has the advantage of generating test problems with tight capacity constraints, which are the most useful problem instances for the computational experiment. However, since the capacity in one period depends on the demand for all previous periods, it is possible that the data generation scheme introduces unintended correlations. This can compromise the Unbiasedness property. Moreover, the random capacity is generated in the first period using a different procedure than is used in all subsequent periods. As a result, the generation scheme fails to satisfy the Consistency property.

In many practical situations, the times at which raw materials become available define time windows for production, as discussed in the context of a multiperiod lot-sizing problem (Brahimi et al. 2006). The testing of Lagrangian heuristics is performed using various parameter specifications for time window density, number of products, demand lumpiness, demand distributions, setup and holding costs, capacity tightness, and minimum and maximum time window lengths. The complex generation scheme satisfies the Consistency, Unbiasedness, Parsimony, and Comparability properties. However, clustering of the time windows is likely to affect problem difficulty and heuristic performance, since it results in resource bottlenecks. This raises concerns about whether the Completeness property can be satisfied without also varying this parameter. Further, for each period, the synthetic data generation procedure ensures that the cumulative aggregate demand is no larger than the cumulative total demand. Unfortunately, the details about how this is accomplished are not given. Thus, the data generation procedure lacks Describability. If the aggregate demand data is generated to fit the cumulative total demand, then this may fail to satisfy the Unbiasedness property. Moreover, if later aggregate demand is treated differently than earlier aggregate demand, then the Consistency property is not satisfied. A data generation process is more consistent if unacceptable aggregate demand streams are discarded and then new streams are generated to replace them. By varying only one parameter at a time, the authors simplify their experiment. However, this also eliminates potentially significant interaction effects (Hall and Posner 2007), which may compromise the Completeness property.

4.3.3 Scheduling

A procedure for the generation of random activity-on-node networks with resource constraints for project scheduling problems is proposed by Demeulemeester et al. (2003). A highly desirable but difficult to achieve feature of such networks is that they are strongly random, i.e., they can be generated at random from the space of all possible networks with specified numbers of nodes and arcs. The proposed procedure, *RanGen*, is the first random network generator that is able to control for two important predictors of project scheduling problem difficulty. It therefore provides an advantage with respect to the Completeness property over previous generators (Patterson 1984). A more specific advantage with respect to the same property is that networks with small densities can be generated, unlike with the previous generators by Kolisch et al. (1995) and Schwindt (1995). Furthermore, to speed up the generation process, some previously published network generators restrict the values of certain parameters that do not affect performance. Unfortunately, these restrictions compromise the randomness of the resulting networks, and violate the Parsimony property. However, *RanGen* satisfies the Parsimony property by avoiding these restrictions. Moreover, this procedure generates project networks that are "as strongly random as possible," given a computation time constraint.

A methodology for data generation for scheduling problems is described by Hall and Posner (2001). The authors observe that release dates become clustered if many release dates are generated within a fixed interval that is independent of the total job processing time. In this situation, the jobs that arrive later do not have active due date restrictions, which fails to satisfy the Consistency property. Regarding due dates, generation schemes in which the earliest due date depends on all the processing times (Hariri and Potts 1983) fail to satisfy the Unbiasedness property. A further concern is that most of the variables may be trivially determined (Potts and Van Wassenhove 1988), in which case the Completeness property is not satisfied. Problems may also arise in the generation of precedence constraint graphs, where as the number of nodes increases, the expected density increases and becomes more varied over different parts of the graph (Potts and Van Wassenhove 1985, van de Velde 1995). This fails to satisfy the Consistency property. Another problem is that the generation of machine speeds from a uniform distribution (Ow 1985) may lead to many machines having similar speeds, which also fails to satisfy the Completeness property. For example, if ten machines have speeds $1, 2, \ldots, 10$, respectively, then the ratio of the speeds of any pair of the last six machines is no larger than two. Improved data generation schemes that resolve these issues are described by Hall and Posner (2001). This work also satisfies all of the generation properties. In particular, the work proposes a precedence constraint generation scheme that both controls the expected density of the precedence constraint graph and also ensures that the expected density is equal across all nodes.

4.3.4 Graphs and Networks

The graph and network problem environment is simpler than many other optimization problems. This makes synthetic data generation easier, and reduces conflicts between the properties. One example of a justifiable data generation scheme is given by Laguna and Rafael (2001). The optimization problem studied involves coloring sparse graphs. Random test graphs are drawn from the class of random graphs with n vertices and edge probability p. Hence, each edge appears in the graph with probability p, independent of whether the other edges appear, which satisfies the Consistency property. The simple generation scheme proposed also satisfies the Unbiasedness, Parsimony, Comparability, Describability and Efficiency properties.

An interesting geometric optimization problem, finding the smallest circle that encloses a set of given circles, is studied by Xu et al. (2003). The centers of the circles in the data set are generated as normally distributed random points, and then the radii are generated as uniformly distributed random numbers. This generation scheme satisfies the Consistency, Unbiasedness, and Parsimony properties. The performance of four fundamentally different algorithms is compared in the computational experiment. These methods are second-order cone optimization, subgradient optimization, quadratic programming, and randomized incremental construction. Given these widely differing solution approaches, it may be appropriate to support the Completeness property by introducing additional variety into the generation scheme. For example, there could be clustering of the given circles, or correlation of the location and size of those circles. It seems likely that including such parameter variations would strengthen the results of the experiment.

A multiperiod capacity expansion problem on networks is considered by Bienstock et al. (2006). The authors use a carefully designed generation scheme that starts with five real-world networks, but varies several parameters randomly, resulting in the generation of 7,750 problem instances. The generation scheme incorporates several features motivated by real-world considerations, including decreasing capacity investment costs and increasing maintenance costs, over time. As a result, the Unbiasedness and Completeness properties are satisfied.

The study of the vertex packing problem by Sherali and Smith (2006) makes use of a formulation in which all rows have to be covered at least once. This is achieved by generating a matrix $A = \{a_{ij}\}$ of random coefficients, where $a_{ij} \in [-1, 1]$. These coefficients are then multiplied by a factor which ensures that $\sum_{j=1}^{n} a_{ij} \geq 1$, for $i = 1, \ldots, m$, and that at least one of these inequalities is tight. This ensures feasibility because $x = (1, \ldots, 1)$ satisfies all constraints. However, the generation scheme fails to satisfy the Unbiasedness and Completeness properties because $x = (1, \ldots, 1)$ is always feasible and because $\sum_{j=1}^{n} a_{ij} = 1$ for at least one $i \in \{1, \ldots, m\}$. A less efficient but also less biased approach would be to generate both feasible and infeasible instances, and then discard the latter. This is an example of a tradeoff between the Efficiency property and the Unbiasedness and Completeness properties.

4.3.5 Routing

One of the most extensively studied optimization problems for which synthetic data is needed is the traveling salesman problem (Arthur and Frendewey 1988). The intractability of this problem (Karp 1972) makes it difficult to find optimal solutions to be used as benchmarks for comparison with heuristic solution procedures. Several possible solutions to this problem are discussed by McGeoch (1996). One possible approach is to generate test problems for which the optimal solution and its value are known by construction. In Arthur and Frendewey (1988), this is achieved by relaxing the subtour elimination constraints in the problem. An optimal solution to the resulting assignment model is used to construct both an optimal solution to the traveling salesman problem and a randomized data set for which that solution is optimal. The authors compare the solution difficulty of their problems with purely random problems, based on their solvability using a standard optimization routine. They conclude that their problems are as hard as purely random ones. However, the proposed procedure fails to satisfy the Completeness property, since the subtour elimination constraints are redundant in all of the data sets they generate. This may lead to unrealistically good performance by some solution procedures, and unrealistically poor performance by others, thus violating the Unbiasedness property. Hence, a detailed computational study would be needed to reveal whether the bias has influenced the results. Moreover, conducting such a study would be approximately as computationally burdensome as solving the intractable optimization problem itself.

An interesting variant known as the black and white traveling salesman problem is studied by Ghiani et al. (2006). For this problem, a tour is feasible only if the number of consecutive white nodes it visits and the tour length between any pair of black nodes both meet prespecified upper bound constraints. Applications of this problem arise in aircraft maintenance. The data generation scheme used by Ghiani et al. (2006) generates the black nodes uniformly among all of the white nodes. It seems unlikely that this is representative of most practical situations. Similar comments can be made about the uniformly generated locations of the pick-up and drop-off locations in a study of the dial-a-ride problem (Cordeau 2006). Consequently, in both research studies, failure to satisfy the Completeness property may be an issue.

Interesting data generation issues arise when sampling from library test data. Recall from the Sect. 4.1 that the library data itself may be flawed. Nonetheless, using this approach, sampling needs to be representative, in order to satisfy the Comparability property. This approach is illustrated by Verweij et al. (2003) to test a sample average approximation method for stochastic routing problems. For each data set, a preliminary graph is chosen from a test bank of graphs. Then, each time a node of the graph is chosen, it is connected to a given number of nodes to which it has not previously been connected. Finally, a source and sink are randomly chosen from among the pairs of nodes that have the maximum value of the minimum number of arcs on any path between them. This approach introduces no additional bias beyond that which may already exist in the library data. It also ensures Completeness through parametric variation of the number of new nodes connected each time.

A computational study of an algorithm for routing in large-scale multicommodity telecommunications networks, discussed by Larsson and Yuan (2004), involves both library and synthetic data. Since different types of data are being used, an important requirement of the generation scheme is to satisfy the Comparability property. The authors address this requirement by generating their random planar networks with a topology that emulates the typical structure of a telecommunications network. The nodes are randomly generated as points in the plane. The arcs are then created in increasing order of Euclidean distance, so that the resulting network is planar. The commodities are defined by randomly chosen node pairs. The arc cost between a pair of nodes is defined by the Euclidean distance between them. This is a reasonable simplification considering the context of the authors' application and experiment.

An important issue in designing a synthetic data generation scheme for vehicle routing problems is integrality of the arc costs. Various procedures for identifying upper and lower bound solutions use truncation, as discussed by Toth and Vigo (2002). Hence, the extent to which the initial data is truncated influences the results. Consequently, the Comparability property may not be satisfied. An example occurs in comparing the results obtained by Fisher (1994) for real-valued Euclidean cost matrices with those obtained by Miller (1995) for Euclidean costs rounded to the nearest integer.

4.3.6 Data Mining

An empirical comparison of the performance of four widely used clustering algorithms, with applications to data mining and knowledge discovery, is considered by Wei et al. (2003). The four algorithms are CLARA (Kaufman and Rousseeuw 1990), CLARANS (Ng and Han 1994), and GAC-R^3 and GAC-RAR$_w$ (Estivill-Castro and Murray 1997). The synthetic data generation scheme by Wei et al. (2003) allows for variations in data size, number of clusters, cluster distributions, cluster asymmetry, and data randomness. Several constraints are imposed on the generation process. First, all clusters are assumed to have the same radius. Also, clusters are assumed to be randomly and evenly divided into two groups, with all clusters in the same group having the same cluster size. Finally, the number of objects in both smaller and larger clusters is specified by a formula. The introduction of these constraints into the data generation process raises several concerns. First, it is unclear whether biases have been introduced, thereby violating the Unbiasedness property. Second, it is unclear whether the constraints introduce variations in the data that are not present in real-world data, thereby violating the Parsimony property. Third, it may be difficult to satisfy the Completeness property because of the restrictions on data variety imposed by the constraints.

An alternative data generation scheme for clustering and outlier analysis is described by Pei and Zaïane (2006). The objective is to generate clustered data points in two or higher dimensional space. One important benefit of this work, relative to the generation scheme by Karyapis et al. (1999), is the detailed explanation of the steps, which documents Describability and promotes replication. The scheme starts with one of several simple probability distributions for the data in each cluster. Next, clusters with different shapes and densities are created using linear transformations, as well as linear equations to control line-based clusters and circle equations to control curve-shaped clusters. The flexibility of this data generation scheme provides Completeness. The generation scheme explicitly generates data sets with five different levels of clustering difficulty and three different levels of outliers, which again satisfies the Completeness property. The Efficiency property is demonstrated through the generation of all data sets with one million points in less than three seconds. The properties of Unbiasedness and Comparability are also satisfied by the generation scheme. It should be noted that the absence of formal constraints creates a data generation environment that is more flexible than many others in the optimization area.

4.3.7 Stochastic Programming

There are two issues in test problem generation for stochastic programming: how to generate the problem data, and how to generate the scenario tree. A popular test problem generator for stochastic programming data is SLP-IOR (Kall and Mayer 1993). This generator includes many options for generating univariate and multivariate, and discrete and continuous distributions. Also, the generator provides the facility to transform between a recourse model, a chance-constrained model, and a deterministic equivalent model, using the same data set. The high level of flexibility provided by SLP-IOR allows for many options in test problem design, and thereby supports the Completeness property. However, there is a lack of published guidelines for choosing from among the many possible problem instances. Perhaps as a consequence, many research studies in stochastic programming use library data instead of generating synthetic data. This avoids some of the problems in satisfying the Describability and Efficiency properties for this application.

There is a more substantial literature on scenario tree generation in multistage stochastic programming. As discussed by Heitsch and Römisch (2005), the main challenge of scenario tree generation is finding a suitable compromise between a good approximation of the underlying probability distribution and the dimension of the stochastic model. The generation approach recommended by the authors minimizes a metric defined on the space of probability distributions. As a result, it satisfies the Unbiasedness property. Conditions are developed by Pennanen (2005) to determine when solutions to a sequence of finer discretizations converge to an optimal solution. This result provides conditions that can be used to satisfy the Unbiasedness property.

Two minimal criteria for scenario tree generation are established by Kaut and Wallace (2003). The first, stability, requires that the same optimal value arises from different scenario trees. The second is that the scenario tree should not introduce any bias relative to the true solution. Further, an approach is proposed for testing scenario generation methods. Several techniques for generating scenario trees that allow interstage dependencies are discussed by Dupačová et al. (2000). These techniques include cluster analysis and importance sampling. A scenario generation heuristic that satisfies the Unbiasedness and Efficiency properties is given by Høyland et al. (2003). The application of nonlinear programming techniques to generate a limited number of discrete outcomes that satisfy prespecified statistical properties is studied by Høyland and Wallace (2001). Also, Dupačová et al. (2003) discuss how a given scenario tree can be reduced to a scenario subset of prespecified cardinality and a probability measure based on this set. Hence, test instances can be generated based on the size of the reduced scenario tree, rather than the original tree. Since many more of the reduced scenario trees can be generated, this supports the Completeness and Efficiency properties. The issue of correlation within the random data (see Sect. 4.3.1) is shown to be potentially significant in an application discussed by Lium et al. (2007). However, no guidelines are provided for incorporating correlation into a data generation scheme for stochastic programming.

Finally, we mention a decomposition-based branch-and-bound algorithm for two-stage stochastic programs that is described by Sherali and Zhu (2007). A computational study is conducted using specific formulations for both the first-stage and the second-stage problems. This raises concerns about whether the Completeness property is being satisfied. In this experiment, there is a trade-off between Completeness and Describability. If the first-stage and the second-stage problem formulations are varied, as the Completeness property suggests, then a description of the data, i.e., the full details of each formulation, is not concise.

4.3.8 Intractable Problems

Knowledge of an optimal solution is frequently required to satisfy the purposes of a given experiment. For many intractable problems, obtaining this information for a set of instances can be very time consuming or even impossible. As a result, some researchers have developed data generation approaches that avoid this difficulty. One approach is to use library data for which optimal solutions are known. Another approach is to generate problem instances with known optimal solutions (Pilcher and Rardin 1992). The proposed procedure is based on obtaining a partial description of the polytope of solutions and then generating random cuts. Once the problem has been generated, both the optimal solution and the form of valid inequalities that is required to solve the problem using cutting planes are known. Starting from a purely random problem and a randomly selected feasible solution, a random sample of tight valid inequalities is selected. This sample, along with dual variable information, is used to compute a cost vector that ensures optimality of the selected solution. This approach is illustrated by generating instances of the asymmetric traveling salesman problem. An extension to the time-dependent traveling salesman problem is described by Vander Wiel and Sahinidis (1995). As observed by Rardin and Uzsoy (2001), the polyhedral approach by Pilcher and Rardin (1992) relies on the availability of problem-specific information, in particular detailed knowledge of the polyhedral structure of the problem. There is empirical evidence that problems generated by this approach are not particularly easy to solve using most heuristics. The potential concerns about failure to satisfy the Unbiasedness and Completeness properties are similar to those in Arthur and Frendewey (1988), as discussed in Sect. 4.3.5. However, the data generation scheme proposed by Pilcher and Rardin (1992) introduces greater randomness into the test problems generated, which mitigates concerns about failure to satisfy the Completeness property.

An important issue with respect to the Completeness property is the generation of data sets with a variety of levels of solution difficulty. This is particularly challenging for an optimization problem that is *NP*-hard (Garey and Johnson 1979). This is because, for many such problems, typical problem instances are easy to solve; however, difficult problem instances occur at critical values of certain problem parameters (Cheeseman et al. 1991). The parameter values where such instances occur are known as *phase transitions*. On one side of the phase transition, there is an underconstrained region where a high density of solutions makes them easy to find. On the other side of the phase transition, there is an overconstrained region with fewer solutions, but where the few solutions that do exist are easy to find (Bailey et al. 2007). Consequently, the hardest problem instances occur between these two regions. Knowledge of where phase transitions occur facilitates the design of a data generation scheme that satisfies the Completeness property. This issue is explored for graph coloring problems in Culberson et al. (1995). The authors identify classes of k-colorable graphs for which various parameters accurately characterize the location of the phase transition. This issue is further explored for the graph bipartitioning problem by Angel and Zissimopoulos (1998). The authors show that the performance of a local search procedure can be predicted from a measure called the autocorrelation coefficient, the maximum distance between any two solutions, and the size of the neighborhood. Angel and Zissimopoulos (2000) extend this idea to describe a hierarchy of *NP*-hard combinatorial optimization problems, based on their autocorrelation coefficients. This hierarchy can be used to evaluate and support the property of Comparability between research studies.

A computational study of a single-phase interior-point algorithm for the monotone complementarity problem is described by Andersen and Ye (1998). One of the features that is tested is how efficiently the algorithm detects an infeasible solution. This raises the issue of how to generate infeasible problem instances. The approach used in Andersen and Ye (1998) sets the right-hand-side value of one of the constraints to be one less than the minimum possible amount attainable if all variables are set to their lower bounds. Instances generated in this way are clearly infeasible. A disadvantage of this scheme, however, is that it makes the right-hand-side value dependent on all of the variable lower bounds, many of which may be redundant in the solution. This fails to satisfy the Parsimony property. Because only one constraint is violated, it also fails to satisfy the Consistency property. A further concern is that the right-hand-side value is always set by using a difference of one. Since the resulting instances are "only just infeasible", this approach generates problem instances for which feasibility may be hardest to detect. However, if the experiment is indeed to estimate how efficiently the algorithm detects an infeasible solution, then the Completeness property suggests that a variety of values for this difference should be used. Moreover, the data generation scheme favors those procedures that efficiently detect minimally infeasible problems, and therefore fails to satisfy the Unbiasedness property.

4.4 Concluding Remarks

We recommend the following general protocol for the generation of synthetic data. First, determine the features of the application and the features of the experiment. Then, use the features of the application and the experiment, along with the generation properties, to develop a data generation scheme. Next, determine the attributes that the data should possess. After the data is generated, measure these attributes to validate the data. We believe that this protocol can be applied quite generally. Possible applications include the various applications of synthetic data generation discussed in the "Introduction".

There are several opportunities for further research related to synthetic data generation in optimization. First, the general protocol which we describe can be specified in greater detail for many classes of optimization problems, following the example by Hall and Posner (2001) for scheduling problems. This will provide a valuable service to researchers, both by standardizing data generation procedures and by eliminating the more problematic practices that currently exist. Also, there are several data generation issues which arise in optimization that pose unusual challenges. Examples include the generation of infeasible problems (Andersen and Ye 1998) and the generation of data for assessing solution quality in stochastic programs (Bayraksan and Morton 2007). These applications of data generation have not been extensively discussed in the literature, and standard procedures need to be developed for them.

Acknowledgements This research is supported by the Summer Fellowship Program of the Fisher College of Business, The Ohio State University, to the first author. Helpful comments were provided by Tito Homem-de-Mello and two anonymous reviewers.

References

Aksoy H, Bayazit M (2000) A model for daily flows of intermittent streams. Hydrological Processes 14:1725–1744

Amini MM, Racer M (1994) A rigorous computational comparison of alternative solution methods for the generalized assignment problem. Management Science

40:868–890

Andersen ED, Ye Y (1998) A computational study of the homogeneous algorithm for large-scale convex optimization. Computational Optimization and Applications 10:243–269

Angel E, Zissimopoulos V (1998) Autocorrelation coefficient for the graph bipartitioning problem. Theoretical Computer Science 191:229–243

Angel E, Zissimopoulos V (2000) On the classification of *NP*-complete problems in terms of their correlation coefficient. Discrete Applied Mathematics 99:261–277

Arthur JL, Frendewey JO (1988) Generating travelling-salesman problems with known optimal tours. Journal of the Operational Research Society 39:153–159

Atamtürk A (2007) Strong formulations of robust mixed 0-1 programming. Mathematical Programming 108:235–250

Bailey DD, Dalmau V, Kolaitis PG (2007) Phase transitions of PP-complete satisfiability problems. Discrete Applied Mathematics 155:1627–1639

Balas E, Martin CH (1980) Pivot and complement - a heuristic for 0-1 programming. Management Science 26:86–96

Balas E, Zemel E (1980) An algorithm for large zero-one knapsack problems. Operations Research 28:1130–1154

Bauer HU, Herrmann M, Villmann T (1999) Neural maps and topographic vector quantization. Neural Networks 12:659–676

Bayraksan G, Morton D (2007) Assessing solution quality in stochastic programs. Mathematical Programming, Series B 108:495–514

Bertsimas D, Natarajan K, Teo CP (2006) Persistence in discrete optimization under demand uncertainty. Mathematical Programming 108:251–274

Beyer K, Goldstein J, Ramakrishnan R (1999) When is “nearest neighbour” meaningful? Database Theory - ICDT ’99 1540:217–235

Bienstock D, Raskina O, Saniee I, Wang Q (2006) Combined network design and multiperiod pricing: Modeling, solution techniques and computation. Operations Research 54:261–276

Bijmolt THA, Wedel M (1999) A comparison of multidimensional scaling methods for perceptual mapping. Journal of Marketing Research 36:277–285

Brahimi N, Dauzère-Pérès S, Najid NM (2006) Capacitated multi-item lot-sizing problems with time windows. Operations Research 54:951–967

Cario MC, Clifford JJ, Hill RR, Yang J, Yang K, Reilly CH (2002) An investigation of the relationship between problem characteristics and algorithm performance: A case study of the GAP. IIE Transactions 34:297–312

Chalmet L, Gelders L (1976) Lagrangean relaxation for a generalized assignment-type problem. North-Holland, Amsterdam, The Netherlands

Cheeseman P, Kanefsky B, Taylor WM (1991) Where the *really* hard problems are. In: Proceedings of IJCAI-91, Morgan Kaufmann, San Mateo, CA, pp 331–337

Chen ZL, Pundoor G (2006) Order assignment and scheduling in a supply chain. Operations Research 54:555–572

Cordeau JJ (2006) A branch-and-cut algorithm for the dial-a-ride problem. Operations Research 54:573–586

Culberson J, Beacham A, Papp D (1995) Hiding our colors. In: Proceedings of the CP ’95 Workshop on Studying and Solving Really Hard Problems, Cassis, France, pp 31–42

Degraeve Z, Schrage L (1997) Should I use a portable generator in an emergency? Working paper, Department of Applied Economic Sciences, Katholieke Universiteit Leuven, Belgium

Demeulemeester E, Vanhoucke M, Herroelen W (2003) RanGen: A random network generator for activity-on-the-node networks. Journal of Scheduling 6:17–38

Dupačová J, Consigli G, Wallace SW (2000) Scenarios for mutistage stochastic programs. Annals of Operations Research 100:25–53

Dupačová J, Gröwe N, Römisch W (2003) Scenario reduction in stochastic programming: An approach using probability metrics. Mathematical Programming, Series A 95:493–511

Estivill-Castro V, Murray AT (1997) Spatial clustering for data mining with generic algorithms. Technical Report FIT-TR-97-10, Faculty of Information Management, Queensland University of Technology

Fischetti M, Lodi A, Martello S, Toth P (2001) A polyhedral approach to simplified crew scheduling and vehicle scheduling problems. Management Science 47:833–850

Fisher ML (1994) Optimal solution of vehicle routing problems using minimum k-trees. Operations Research 42:626–642

Freed JA (2000) Conceptual comparison of two computer models of corpuscle sectioning and of two algorithms for correction of ploidy measurements in tissue sections. Analytical and Quantitative Cytology and Histology 22:17–25

Frenje L, Juhlin C (1998) Scattering of seismic waves simulated by finite difference modelling in random media: Application to the Gravberg-1 well. Sweden Tectonophysics 293:61–68

Garey MR, Johnson DS (1979) Computers and Intractability: a Guide to the Theory of NP-Completeness. W.H. Freeman, San Francisco, CA

Gelius LJ, Westerdahl H (1997) Seismic noise modelling. Journal of Seismic Exploration 6:351–366

Ghiani G, Laporte G, Semet F (2006) The black and white traveling salesman problem. Operations Research 54:366–378

Gonzalez J, Gutierrez R (1999) Direct motion estimation from a range scan sequence. Journal of Robotic Systems 16:73–80

Goutte C (2000) Extraction of the relevant delays in temporal modelling. IEEE Transactions on Signal Processing 48:1787–1795

Grate JW, Wise BM, Abraham MH (1999) Method for unknown vapor characterization and classification using a multivariate sorption detector. Analytical Chemistry 71:4544–4553

Guignard M, Rosenwein MB (1989) An improved dual based algorithm for the generalized knapsack problem. Operations Research 37:658–663

Hadjar A, Marcotte O, Soumis F (2006) A branch-and-cut algorithm for the multiple depot vehicle scheduling problem. Operations Research 54:130–149

Hall NG, Posner ME (2001) Generating experimental data for computational testing with machine scheduling applications. Operations Research 49:854–865

Hall NG, Posner ME (2007) Performance prediction and preselection for optimization procedures. Operations Research 55:703–716

Hariri AM, Potts CN (1983) An algorithm for single machine sequencing with release dates to minimize total weighted completion time. Discrete Applied Mathematics 5:99–109

Hays WL (1973) Statistics for the Social Sciences, 2nd edn. Holt, Rinehart and Winston, Inc., New York, NY

Heitsch H, Römisch W (2005) Generation of multivariate scenario trees to model stochasticity in power management. In: Power Tech, IEEE Russia, pp 1–7

Herique A (1999) Radio wave back-propogating in polar coordinates: A linear filter in the time-frequency angle-frequency domain. Radio Science 34:509–519

Hill RR, Reilly CH (1994) Composition for multivariate random variables. In: Proceedings, 1994 Winter Simulation Conference, Institute of Electrical and Electronics Engineers, Orlando, FL, pp 332–342

Hill RR, Reilly CH (2000) The effects of coefficient correlation structure in two-dimensional knapsack problems on solution procedure. Management Science 46:302–317

Ho TK, Baird HS (1997) Large-scale simulation studies in image pattern recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence 19:1067–1079

Hooker JN (1994) Needed: An empirical science of algorithms. Operations Research 42:201–212

Hooker JN (1995) Testing heuristics: We have it all wrong. Journal of Heuristics 1:33–42

Hooshyar MA, Lam TH, Razavy M (2000) Inverse problem of the wave equation and the Schwinger approximation. Journal of the Acoustical Society of America 107:404–413

Høyland K, Wallace SW (2001) Generating scenario trees for multstage decision problems. Management Science 47:295–307

Høyland K, Kaut M, Wallace SW (2003) A heuristic for moment-matching scenario generation. Annals of Operations Research 24:169–185

Iman RL, Conover WJ (1982) A distribution-free approach to inducing rank correlation among input variables. Communications in Statistics: Simulation and Computing B11:311–334

John TC (1989) Tradeoff solutions in single machine production scheduling for minimizing flow time and maximum penalty. Computers & Operations Research 16:471–479

Kadlec RH (2000) The inadequacy of first-order treatment wetland models. Ecological Engineering 15:105–119

Kall P, Mayer J (1993) SLP-IOR: On the design of a workbench for testing SLP codes. Revista Investigación Operacional 14:148–161

Karp RM (1972) Reducibility among combinatorial problems. In: Complexity of Computer Computations, Plenum, New York, NY, pp 85–103

Karyapis G, Han EH, Kumar V (1999) CHAMELEON: A hierarchical clustering algorithm using dynamic modeling. IEEE Computer 32:68–75

Kaufman L, Rousseeuw PJ (1990) Finding Groups in Data: An Introduction to Cluster Analysis. Wiley, New York, NY

Kaut M, Wallace SW (2003) Evaluation of scenario-generation methods for stochastic programming. Working paper, Molde University College, Norway

Koehler JR, Owen AB (1996) Computer experiments. In: Ghosh S, Rao C (eds) Handbook of Statistics, vol 13, Elsevier Science, New York, NY, pp 261–308

Kolisch R, Sprecher A, Drexl A (1995) Characterization and generation of a general class of resource-constrained project scheduling problems. Management Science 41:1693–1703

Krieger AM, Green PE (1999) A cautionary note on using internal cross validation to select the number of clusters. Psychometrika 64:341–353

Laguna M, Rafael M (2001) A GRASP for coloring sparse graphs. Computational Optimization and Applications 19:165–178

Larsson T, Yuan D (2004) An augmented Lagrangian algorithm for large scale multicommodity routing. Computational Optimization and Applications 27:187–215

Law AM, Kelton WD (1991) Simulation Modeling and Analysis, 2nd edn. McGraw-Hill, New York, NY

Linderoth J, Shapiro A, Wright S (2006) The empirical behavior of sampling methods for stochastic programming. Annals of Operations Research 142:215–241

Lium AG, Crainic TG, Wallace SW (2007) Correlations in stochastic programming: A case from stochastic service network design. Revista Investigación Operacional 24:161–179

Lu Z, Wyss M, Pulpan H (1997) Details of stress directions in the Alaska subduction zone from fault plane solutions. Journal of Geophysical Research-Solid Earth 102:5385–5402

Martello S, Toth P (1979) The 0-1 knapsack problem. In: Christofides N, Mingozzi A, Toth P, Sandi C (eds) Combinatorial Optimization, Wiley, New York, NY, pp 237–279

Martello S, Toth P (1981) An algorithm for the generalized assignment problem. In: Brans JP (ed) Operational Research '81, North-Holland, Amsterdam, The Netherlands, pp 589–603

Martello S, Toth P (1988) A new algorithm for the 0-1 knapsack problem. Management Science 34:633–644

Martello S, Toth P (1997) Upper bounds and algorithms for hard 0-1 knapsack problems. Operations Research 45:768–778

McGeoch CC (1996) Towards an experimental method for algorithm simulation. INFORMS Journal on Computing 8:1–15

McIntosh SW, Charbonneau P, Brown JC (2000) Preconditioning the differential emission measure (T-e) inverse problem. Astrophysics Journal 529:1115–1130

Miller DL (1995) A matching based exact algorithm for capacitated vehicle routing problems. ORSA Journal on Computing 7:1–9

Munizaga MA, Heydecker BG, Ortuzar JD (2000) Representation of heteroskedasticity in discrete choice models. Transportation Research B - Methodology 34:219–240

Ng R, Han J (1994) Efficient and effective clustering methods for spatial data mining. In: Proceedings of International Conference on Very Large Data Bases, Santiago, Chile, pp 144–155

Ow P (1985) Focused scheduling in proportionate flowshops. Management Science 31:852–869

Palmer CR, Faloutsos C (2000) Density biased sampling: An improved method for data mining and clustering. SIGMOD Record 29:82–92

Pan Y, Shi L (2007) On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems. Mathematical Programming 110:543–559

Patterson JH (1984) A comparison of exact procedures for solving the multiple-constrained resource project scheduling problem. Management Science 30:854–867

Pearson WR, Robins G, Zhang TT (1999) Generalized neighbor-joining: more reliable phylogenetic tree reconstruction. Molecular Biology and Evolution 16:806–816

Pei Y, Zaïane O (2006) A synthetic data generator for clustering and outlier analysis. In: Technical report TR06-15, University of Alberta, Edmonton, Alberta

Pennanen T (2005) Epi-convergent discretizations of multistage stochastic programs. Mathematics of Operations Research 30:245–256

Pilcher MG, Rardin RL (1992) Partial polyhedral description and generation of discrete optimization problems with known optima. Naval Research Logistics 39:839–858

Potts CN, Van Wassenhove LN (1985) A Lagrangean based branch and bound algorithm for single machine sequencing with precedence constraints to minimize total weighted completion time. Management Science 31:1300–1311

Potts CN, Van Wassenhove LN (1988) Algorithms for scheduling a single machine to minimize the weighted number of late jobs. Management Science 34:843–858

Potts CN, Van Wassenhove LN (1992) Single machine scheduling to minimize total late work. Operations Research 40:586–595

Qin G, Jing BY (2000) Asymptotic properties for estimation of partial linear models with censored data. Journal of Statistical Planning and Inference 84:95–110

Racer M, Amini MM (1994) A robust heuristic for the generalized assignment problem. Annals of Operations Research 50:487–503

Rardin RL, Uzsoy R (2001) Experimental evaluation of heuristic optimization algorithms: A tutorial. Journal of Heuristics 7:261–304

Ray KS, Ghoshal J (2000) Neuro-genetic approach to multidimensional fuzzy reasoning for pattern classification. Fuzzy Sets and Systems 112:449–483

Reiter JP (2002) Satisfying disclosure restrictions with synthetic data sets. Journal of Official Statistics 18:531–543

Romeijn HE, Morales DR (2001a) Generating experimental data for the generalized assignment problem. Operations Research 49:866–878

Romeijn HE, Morales DR (2001b) A probabilistic analysis of the multi-period single-sourcing problem. Discrete Applied Mathematics 112:301–328

Ross GT, Soland RM (1975) A branch and bound algorithm for the generalized assignment problem. Mathematical Programming 8:91–103

Ross GT, Soland RM (1977) Modeling facility location problems as generalized assignment problems. Management Science 24:345–357

Roversi P, Irwin JJ, Bricogne G (1998) Accurate charge density studies as an extension of Bayesian crystal structure determination. Acta Crystallographica Section A 54:971–996

Ruchala KJ, Olivera GH, Schloesser EA (1999) Megavoltage CT on a tomotherapy system. Physics in Medicine and Biology 44:2597–2621

Rushmeier RA, Nemhauser GL (1993) Experiments with parallel branch-and-bound algorithms for the set covering problem. Operations Research Letters 13:277–285

Schaffer C (1994) A conservation law for generalization performance. In: International Conference on Machine Learning, Morgan Kaufmann, San Francisco, CA, pp 259–265

Schena G, Chiaruttini C (2000) A stereologically posed mass balance for calculating the distributed efficiency of particle separation systems. International Journal of Mineral Processing 59:149–162

Schwindt C (1995) A new problem generator for different resource-constrained project scheduling problems with minimal and maximal time lags. WIOR-Report-449, Institut für Wirtschaftstheorie und Operations Research, University of Karlsruhe

Shen L, Shen H, Cheng L (1999) New algorithms for efficient mining of association rules. Information Sciences 118:251–268

Sherali HD, Smith JC (2006) A polyhedral study of the generalized vertex cover problem. Mathematical Programming 107:367–390

Sherali HD, Zhu X (2007) On solving discrete two-stage stochastic programs having mixed-integer first- and second-stage variables. Mathematical Programming 105:597–616

Smith-Miles KA (2008) Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Computing Surveys 41:6.1–6.25

Smith-Miles KA, James RJW, Giffin JW, Tu Y (2009) A knowledge discovery approach to understanding relationships between scheduling problem structure and heuristic performance. In: Learning and Intelligent OptimizatioN Conference (LION 3), Trento, Italy

Toth P, Vigo D (2002) Models, relaxations and exact approaches for the capacitated vehicle routing problem. Discrete Applied Mathematics 123:487–512

Trick MA (1992) A linear relaxation heuristic for the generalized assignment problem. Naval Research Logistics 39:137–151

Uma RN, Wein J (1998) On the relationship between combinatorial and LP-based approaches to *NP*-hard scheduling problems. Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science 1412:394–408

Vander Wiel RJ, Sahinidis NV (1995) Heuristic bounds and test problem generation for the time-dependent traveling salesman problem. Transportation Science 29:167–183

van de Velde SL (1995) Dual decomposition of a single-machine scheduling problem. Mathematical Programming 69:413–428

Verweij B, Ahmed S, Kleywegt A, Nemhauser G, Shapiro A (2003) The sample average approximation method applied to stochastic routing problems: A computational study. Computational Optimization and Applications 24:289–333

Wei CP, Lee YH, Hsu CM (2003) Empirical comparison of fast partitioning-based clustering algorithms for large data sets. Expert Systems with Applications 24:351–363

Wilson RC, Hancock ER (2000) Bias variance analysis for controlling adaptive surface meshes. Computer Vision and Image Understanding 77:25–47

Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1:67–82

Wu D, Golbasi H (2004) Multi-item, multi-facility supply chain planning: Models, complexities and algorithms. Computational Optimization and Applications 28:325–356

Xu S, Freund RM, Sun J (2003) Solution methodologies for the smallest enclosing circle problem. Computational Optimization and Applications 25:283–292

Yaman H, Karaşan OE, Pinar MÇ (2007) Restricted robust uniform matroid maximization under interval uncertainty. Mathematical Programming 110:431–441

Yuval (2000) Neural network training for prediction of climatological time series, regularized by minimization of the generalized cross validation function. Monthly Weather Review 128:1456–1473

Chapter 5
The Attainment-Function Approach to Stochastic Multiobjective Optimizer Assessment and Comparison

Viviane Grunert da Fonseca and Carlos M. Fonseca

Abstract This chapter presents the attainment-function approach to the assessment and comparison of stochastic multiobjective optimizer (MO) performance. Since the random outcomes of stochastic MOs, such as multiobjective evolutionary algorithms, are sets of nondominated solutions, analyzing their performance is challenging in that it involves studying the distribution of those sets. The attainment function, so named because it indicates the probability of an MO attaining an arbitrary goal, is related to results from random closed-set theory which cast it as a kind of mean for the distribution of the optimizer outcomes in objective space. Higher-order versions of the attainment function can also address other aspects of this distribution, and may even lead to a full distributional characterization. This approach to the experimental assessment and comparison of MO performance is based on statistical inference methodology, in particular, estimation and hypothesis testing.

5.1 Introduction

The importance of stochastic optimizers, such as evolutionary algorithms, simulated annealing, and particle-swarm optimization algorithms, is well recognized in many practical applications arising in a broad range of scientific and engineering domains. Solving an optimization problem consists of determining element(s) of a "decision space" which are optimal under a scalar or vector-valued objective function, meaning that the corresponding images in the space of objective function values (the

Viviane Grunert da Fonseca
INUAF – Instituto Superior D. Afonso III, Loulé, Portugal, e-mail: viviane.grunert@vodafone.pt
CEG-IST – Centre for Management Studies, Instituto Superior Técnico, Lisbon, Portugal

Carlos M. Fonseca
Department of Electronic Engineering and Informatics, Faculty of Science and Technology, Universidade do Algarve, Faro, Portugal, e-mail: cmfonsec@ualg.pt
CEG-IST – Centre for Management Studies, Instituto Superior Técnico, Lisbon, Portugal

"objective space") are minimal or maximal elements (Taylor 1999, p. 131) of the objective function image set.

In *single*-objective optimization problems, the objective space is usually considered to be the real line, $\mathbb{R}$, and optimizers typically produce single solutions whose one-point images in this space try to approximate the unknown optimal value (minimum or maximum) of the objective function. Most real-world problems, however, involve *several*, possibly conflicting, objectives to be optimized simultaneously. Considering d objectives, the objective space is now $\mathbb{R}^d$, and the optimizer needs to approximate the unknown set of *Pareto-optima* of a vector-valued function, the image of which in objective space is known as the *Pareto-optimal front* (see, for example, Ben-Tal 1980 for a formal definition of Pareto optimum). Thus, rather than a single point, the outcome of a multiobjective optimizer is typically a set of *nondominated* solutions whose images in the objective space $\mathbb{R}^d$ approximate the unknown Pareto-optimal front of the problem.

A vast number of different optimizers have been developed to date, and it becomes more and more challenging to identify, among so many alternative, competing algorithms, which one performs "best" on a given (class of) optimization problem(s). In general, optimizer performance implies a trade-off between the quality of the solutions produced and the computational effort needed to produce them (Grunert da Fonseca et al. 2001). When optimizers are stochastic, both the solutions and the computational effort are *random*, and the study of optimizer performance requires the description of the corresponding *probability distributions* in objective space. In other words, given a particular MO and a d-objective problem, one needs to consider the distribution of a set of nondominated points in $\mathbb{R}^d$, when focusing on solution quality only, or in $\mathbb{R}^{d+1}$, if the computational effort is also of interest. Note that, from an optimizer performance point of view, the computational effort can be seen as just another objective (to be minimized).

In the literature, many attempts have been made to describe these set distributions through summary measures, or quality indicators, which assign a real value to the realized outcome of a single run of an optimizer (Zitzler et al. 2003). To acknowledge the stochastic nature of such outcomes, these values are usually averaged over several optimization runs. The attainment-function approach, on the other hand, is formulated on a *functional* basis, and recognizes the set distribution of optimizer outcomes as a whole. Partial aspects of this distribution are then selected to be considered, following criteria from statistical inference theory.

After explaining the important role of statistics in MO performance assessment (Sect. 5.2), the attainment-function approach will be presented from the ground up, i.e., from the identification of the MO outcomes as random nondominated point sets with a particular type of probability distribution (Sect. 5.3), through the theoretical description of (aspects of) this distribution involving probability theory (Sects. 5.4 and 5.5), to the empirical description and comparison of (aspects of) this distribution using statistical inference based on multiple and independent optimization runs (Sects. 5.6 and 5.7). The chapter concludes with a discussion of the current status of the approach and some perspectives for future work (Sect. 5.8). Without loss of generality, *minimization* problems will be considered throughout.

5.2 Statistics and the Attainment-Function Approach

The attainment-function approach to the performance assessment and comparison of stochastic multiobjective optimizers rigorously follows concepts and ideas from *statistical* inference theory and methodology.

In general, statistical inference is concerned with three different kinds of *inference procedures*: (point) estimators, confidence intervals, and hypothesis tests. Only two of them will be considered in this chapter: *estimators* for the assessment of MO performance (Sect. 5.6) and *hypothesis tests* for the comparison of MO performance (Sect. 5.7). While the general methodology of both types of inference procedures is explained in Appendix A, the discussion in the present section will focus on the logical link between stochastic optimizers and statistics, as provided by the attainment-function approach.

5.2.1 Stochastic Optimizers as Statistical Estimators

Stochastic multiobjective optimizers can be seen as *statistical estimators* for the unknown, true Pareto-optimal front of a vector function in objective space $\mathbb{R}^d$, $d \geq 1$ (Grunert da Fonseca et al. 2001).

Like all statistical inference procedures, estimators are generally aimed at describing (an aspect of) the unknown probability distribution of some population with respect to a random variable/vector of interest. For example, one might be interested in estimating the mean consumption of fuel in a certain brand of vehicles registered in a given European country. Then, the collection of all such vehicles constitutes the population, and its univariate distribution with respect to the random variable "fuel consumption" is considered. The aspect of this distribution to be described through estimation is the unknown mean.

Statistical estimators use information from (the variable/vector values of) a set of population elements, which are usually selected randomly and independently from each other, and are identically distributed like the unknown population distribution. In other words, statistical estimators are *statistics*, in the sense that they are functions of a *random sample* from the population, and they possess a particular random (sampling) distribution. Thus, given a simple (i.e., independently drawn) random sample of n vehicles of the (much) larger vehicle population, an estimator for the mean fuel consumption could be the arithmetic mean of all fuel consumption values of that sample (at a certain time of study). This particular estimator is the well-known sample average, and a classical result in statistics is the "Central Limit Theorem," stating that the sampling distribution of the sample average, based on a simple random sample from a population with existing mean and variance, converges to a normal distribution as n increases (Mood et al. 1974, p. 195 and Appendix A).

The simple fuel-consumption example given above could be modified in several ways. For instance, two or more random variables might be of interest simultaneously, such as “fuel consumption,” “production price,” *etc.*, in which case the (joint) population distribution would be multivariate. The aspect to be estimated from such a population distribution might still be the mean vector, but it could also be any other aspect of interest, leading to different estimators. Moreover, it is not uncommon in statistical inference to consider random samples of dependent elements. For example, Markov Chain Monte Carlo methods are often preferred to conventional (independent) Monte Carlo simulation in the estimation of multidimensional integrals (Gilks et al. 1996).

In a stochastic multiobjective optimization context, the set of all possible solutions in decision space may be seen as the population, with the (multivariate) population distribution being taken with respect to the vector of the corresponding objective-function values. The set of solutions actually evaluated in an optimization run may be understood as a random, though generally *not* independently drawn, sample. In fact, independent random sampling corresponds to the simplest stochastic optimizer of all, i.e., pure random search, whereas most practical optimizers perform sampling by means of a stochastic process (typically, a Markov chain).

The optimizer aims to approximate the unknown (true) Pareto-optimal front of a multiobjective optimization problem. Therefore, when considering minimization, the aspect of the population distribution to be estimated is its lower boundary of support. The estimate is obtained by determining the set of non-dominated solutions in the sample of solutions evaluated in one optimization run, and returning the corresponding image in objective space. Thus, a stochastic MO holds a random sampling distribution which depends both on the random sampling mechanism it implements and on the distribution of the solutions in objective space (the population distribution), through a specific *set*-valued function of the sample.

Unlike the asymptotic distribution of the sample average, optimizer sampling distributions (exact or asymptotic) are generally not known. In the following, these set distributions will be referred to as *optimizer outcome distributions*, and will be the focus of the subsequent considerations.

5.2.2 Optimizer Performance as Statistical Estimator Performance

The performance of stochastic multiobjective optimizers can be studied using criteria usually considered for frequency-based statistical estimators (Grunert da Fonseca et al. 2001).

The classical, frequency-based school of inference postulates that the process of sampling from the population and subsequent estimation can be repeated arbitrarily often. Thus, performance criteria which reflect the “statistical error” of an estimator can be defined by referring to the estimator's random (sampling) distribution. Good performance of point estimators, for example, implies that repeated estimation results tend to be *close* to the unknown estimand (i.e., the aspect to be estimated from the population distribution) both in terms of *location* and *spread* of the sampling distribution (Mood et al. 1974, p. 289). A second performance criterion for point estimators may relate to the *overall* sampling distribution, in the sense that its form is (approximately) known and easy to deal with, as it happens, for example, with the sample average and the normal distribution (see page 105).

The classical idea of considering *repeated* estimation results for the purpose of estimator performance assessment is particularly suitable for stochastic optimizers, as repeated optimization results (outcomes) may be easily obtained through *multiple* runs of the optimizer. Hence, it is perfectly justified to express the randomness of optimizer outcomes in terms of the optimizer's sampling distribution, and to refer precisely to this *optimizer outcome distribution* when discussing optimizer performance.

Nevertheless, several difficulties arise in the context of multiobjective optimization. Rather than single values in $\mathbb{R}$, as produced by point estimators, MO outcomes are sets of points in $\mathbb{R}^d$ with a certain nondominance restriction. In other words, instead of studying univariate sampling distributions which can be defined in terms of (univariate) cumulative distribution functions, one needs to consider particular set distributions, the full characterization of which is less obvious, and requires the use of *random closed-set theory*. Furthermore, the question of how to define a suitable notion of "closeness" to the estimand of a multiobjective optimizer arises. Rather than evaluating proximity to a single point in $\mathbb{R}$, as in the case of point estimators, one needs to agree about a certain "distance" to some Pareto-optimal front (the estimand), which may go from a single point to a hypersurface in $\mathbb{R}^d$. Finally, since the set distribution of optimization outcomes is bounded below, and the unknown estimand is its lower boundary of support, some degree of skewness is desired. However, this problem is not trivial to address, since skewness measures for univariate distributions are certainly not appropriate. The following sections will try to give answers to the above questions.

In Sect. 5.3, the stochastic outcomes of single MO runs will be formally identified as so-called *random nondominated point (RNP) sets*, which also possess an alternative representation as *attained sets*. Their complete distributional characterization, as a generalization of the cumulative distribution function, will be explained in Sect. 5.4 through the definition of the k-th-order attainment function. Univariate ideas of closeness to the estimand will be extended to (multivariate) RNP sets in Sect. 5.5, by considering the first-order attainment function for the purpose of *location*, by defining the *variance function* to explain *spread* (accounting for variability across multiple optimization runs), and, finally, by considering higher-order attainment functions to assess *inter-point dependence structures*. This last aspect becomes important due to the particular set-character of MO outcomes.

5.2.3 Performance Assessment via Estimation and Hypothesis Testing

For a *particular* optimization problem, multiobjective optimizer performance can be assessed and compared *empirically* through statistical estimation and hypothesis testing using empirical (higher-order) attainment functions (Fonseca et al. 2005).

The study and comparison of estimator performance on a purely *theoretical* basis is usually satisfactory only when certain parametric assumptions (sometimes including actual parameter values) can be made about the population distribution and, therefore, about the estimator's sampling distribution. Obviously, the less information about this distribution is available, the more general and vague any theoretical knowledge about the corresponding estimator performance becomes. Especially in nonparametric situations, where very little is assumed, the *empirical* study of estimator performance becomes important.

Hüsler et al. (2003) justified a Weibull distribution for the sampling distribution of *single*-objective random search, and determined the corresponding parameter values for some given optimization problems. The authors suggested that this parametric assumption could be used in describing optimizer performance. However, this has been one of the few attempts to find theoretical (semi)parametric models for single-objective optimizer outcome distributions, and extensions to the multiobjective case are not straightforward. Hence, to describe MO performance, the easiest workaround is to assume a less informative nonparametric situation, and to determine (higher-order) attainment functions and related measures completely in an *empirical* way. In other words, the (higher-order) theoretical attainment function is entirely unknown, and needs to be *estimated* as a whole using a random sample of independent and identically distributed MO solution sets obtained through multiple optimization runs.[1] Additionally, hypothesis tests based on empirical attainment functions can be used to (eventually) infer that two or more optimizers are different in performance, or that a given optimizer does not perform better than, or as well as, another, while maintaining control over the statistical error associated with these conclusions.

Sections 5.6 and 5.7 are devoted to statistical estimation and testing with (higher-order) attainment functions.

[1] The quality (i.e., performance) of the empirical (higher-order) attainment function, *again* seen as a statistical estimator, can be assessed according to the general frequency-based performance criteria mentioned above (this time through repeated samples of multiple MO solution sets).

5.3 Multiobjective Optimizer Outcomes

The outcome of a stochastic multiobjective optimizer is considered to be the image in objective space of the set of solutions generated in one optimization run in a given amount of time, where time may be measured in terms of number of iterations, number of function evaluations, CPU time, elapsed time, *etc.* A precise mathematical definition of an MO outcome as a particular type of random point set is given next.

5.3.1 Random Nondominated Point Sets

For a d-objective optimization problem, the elements of an optimizer outcome set are objective vectors, defined as points in the objective space $\mathbb{R}^d$, which are *nondominated* in the sense that no member of that set is considered to be better (i.e., smaller, without loss of generality) than any other member.

Definition 5.1. (Nondominance) The points $p_1, p_2, \ldots \in \mathbb{R}^d$ are said to be nondominated under the Pareto-order relation if there does not exist any pair $(i, j) \in \mathbb{N}^2$, $i \neq j$, for which $p_i \leq p_j$.

The outcome sets of stochastic d-objective optimizers are *random* because their elements are random vectors in $\mathbb{R}^d$ and because their cardinality is random, though finite with probability one. Statistically, MO outcomes are *random nondominated point sets* (RNP sets):

Definition 5.2. (Random nondominated point set, RNP set) A random point set

$$\mathcal{X} = \{X_1, \ldots, X_M \in \mathbb{R}^d : \ \Pr\{X_i \leq X_j\} = 0, \ i \neq j\},$$

where both the cardinality M and the elements X_i are random, $i = 1, \ldots, M$, and where $\Pr\{0 \leq M < \infty\} = 1$, is called a random nondominated point set (Fonseca et al. 2005).

Note that the nondominance condition introduces dependence among the elements of an RNP set, and that the empty set $\emptyset$ may be a realization of $\mathcal{X}$ when a given optimization run produces *no* solution at all.

5.3.2 Alternative View: The Attained Set

As an *optimizer outcome distribution*, the distribution of an RNP set is of interest for the description of multiobjective optimizer performance (Sects. 5.2.1 and 5.2.2). In principle, this distribution could be characterized with theory for finite multidimensional stochastic point processes (Daley and Vere-Jones 1988, Chap. 5). However,

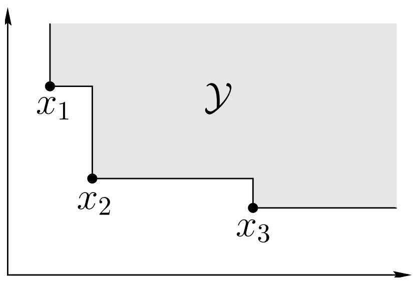


Fig. 5.1: RNP set $\mathcal{X}$ with nondominated realizations x_1, x_2, and x_3 and the attained set $\mathcal{Y}$, here as a realization with $M = 3$ (from Grunert da Fonseca et al. 2001)

in terms of the mathematics involved, it seems to be preferable to pursue an alternative approach by taking advantage of the particular nondominance condition. The so-called *attained set*, or "dominated space" (Zitzler 1999, p. 43), has a distribution which is equivalent to that of the corresponding RNP set.

Definition 5.3. (Attained set) The random set

$$\begin{aligned}\mathcal{Y} &= \{y \in \mathbb{R}^d \mid X_1 \leq y \ \vee \ X_2 \leq y \ \vee \ldots \vee \ X_M \leq y\} \\ &= \{y \in \mathbb{R}^d \mid \mathcal{X} \trianglelefteq y\}\end{aligned}$$

is the set of all goals $y \in \mathbb{R}^d$ attained by the RNP set $\mathcal{X}$ (Fonseca et al. 2005).

Unless it is the empty set, a realization of the attained set $\mathcal{Y}$, obtained after one optimization run, is composed of all, infinitely many, points in objective space which are greater (i.e., worse) than or equal to the points of the corresponding realization of $\mathcal{X}$ as illustrated in Fig. 5.1. Thus, a realization of the attained set $\mathcal{Y}$ is a closed and unbounded subset in $\mathbb{R}^d$, whereas $\mathcal{Y}$ itself is called a random closed set. The distribution of $\mathcal{Y}$ can be described using random closed-set theory (Harding and Kendall 1974, Matheron 1975, Goutsias 1998).

5.4 Multiobjective Optimizer Performance

The performance of a multiobjective optimizer can be discussed by referring to the distribution of its outcome RNP set or, equivalently, by studying the distribution of the corresponding attained set. A complete distributional description of the latter, as a particular type of random closed set, automatically leads to a total (distributional) characterization of the former, and vice versa.

5.4.1 *Distribution of a General Random Closed Set: The Capacity Functional*

A characterization of the distribution of a general random closed set $\mathcal{W} \subset \mathrm{I\!R}^d$ is provided by the capacity functional (or hitting function), which is based on so-called "hit-or-miss events" indicating the nonempty intersection of the random closed set with some compact (closed and bounded) test set K. In mathematical terminology, the capacity functional of $\mathcal{W}$ is defined as

$$T_{\mathcal{W}}(K) = \Pr\{\mathcal{W} \cap K \neq \emptyset\},$$

where K is a compact subset of $\mathrm{I\!R}^d$ (Goutsias 1998, p. 5). In other words, the value of the capacity functional $T_{\mathcal{W}}(\cdot)$ on a given deterministic test set K indicates the probability of the random closed set $\mathcal{W}$ hitting the set K. The knowledge of $\Pr\{\mathcal{W} \cap K \neq \emptyset\}$ for *all* compact subsets K uniquely identifies the distribution of $\mathcal{W}$ (the Choquet-Kendall-Matheron Theorem, see Goutsias 1998, p. 7).

In the context of multiobjective optimization, this rather demanding characterization with the collection of *all* possible compact test sets K of $\mathrm{I\!R}^d$ is not actually needed. On the one hand, the knowledge of $T_{\mathcal{Y}}(\cdot)$ for some test set K^* immediately leads to the knowledge of $T_{\mathcal{Y}}(\cdot)$ for all other test sets which share with K^* the same upper boundary. On the other hand, the discrete nature of $\mathcal{X}$ implies a step-like lower boundary of the attained set $\mathcal{Y}$ (as in Fig. 5.1), which allows a full distributional characterization with $T_{\mathcal{Y}}(\cdot)$ based on a much *smaller* class of compact test sets (Grunert da Fonseca and Fonseca 2004).

5.4.2 *Distribution of a Random Nondominated Point Set: The k-th-Order Attainment Function*

The distribution of an RNP set $\mathcal{X}$ holding a random cardinality M with support on $\{0, 1, \ldots, m^*\}$ (i.e., the corresponding optimizer produces up to m^* nondominated solutions per run), and that of the corresponding attained set $\mathcal{Y}$, can be uniquely characterized through the capacity functional of $\mathcal{Y}$ defined over the collection of all deterministic point test-sets $\{z_1, \ldots, z_{m^*}\} \subset \mathrm{I\!R}^d$.[2] That is, the performance of such a multiobjective optimizer can be characterized by

$$\begin{aligned} T_{\mathcal{Y}}(\{z_1, \ldots, z_{m^*}\}) &= \Pr\{\mathcal{Y} \cap \{z_1, \ldots, z_{m^*}\} \neq \emptyset\} \\ &= \Pr\Big\{\big(\mathcal{Y} \cap \{z_1\} \neq \emptyset\big) \;\vee\; \ldots \;\vee\; \big(\mathcal{Y} \cap \{z_{m^*}\} \neq \emptyset\big)\Big\} \end{aligned}$$

[2] Strictly speaking, the collection of compact test sets to be considered for characterization could even be smaller, i.e., only contain every set of the kind $\{z_1, \ldots, z_\ell\}$, where $\ell = 1, \ldots, m^*$ and all $z_i \in \mathrm{I\!R}^d$ are nondominated, $i = 1, \ldots, \ell$ (Grunert da Fonseca and Fonseca 2004, Theorem 1). This characterization, however, is less practical in the present context.

for all z_i in objective space $\mathbb{R}^d$, $i = 1, \ldots, m^*$ (see Grunert da Fonseca and Fonseca 2004, Lemma 1, for the general case $m^* \geq 1$, and Grunert da Fonseca and Fonseca 2002, for the case $m^* = 1$). As a consequence, it can be shown that the distribution of $\mathcal{X}$ and $\mathcal{Y}$ can also be completely described by the probabilities

$$\Pr\left\{(\mathcal{Y}\cap\{z_1\} \neq \emptyset) \wedge \ldots \wedge (\mathcal{Y}\cap\{z_{m^*}\} \neq \emptyset)\right\} = \Pr\left\{\mathcal{X} \trianglelefteq z_1 \wedge \ldots \wedge \mathcal{X} \trianglelefteq z_{m^*}\right\}$$

for all $z_i \in \mathbb{R}^d$, $i = 1, \ldots, m^*$ (Grunert da Fonseca and Fonseca 2004, Lemma 2). This is of particular relevance for the interpretation of MO performance, as it is associated with the notion of simultaneously attaining the m^* objective vectors (goals) $z_1, \ldots, z_{m^*}$ in objective space $\mathbb{R}^d$. In general, it motivates the definition of the *k-th-order attainment function*:

Definition 5.4. (k-th-order attainment function) The function defined as $\alpha_{\mathcal{X}}^{(k)}$: $\mathbb{R}^{d\times k} \longrightarrow [0, 1]$ with

$$\alpha_{\mathcal{X}}^{(k)}(z_1, \ldots, z_k) = \Pr\left\{\mathcal{X} \trianglelefteq z_1 \;\wedge\; \ldots \;\wedge\; \mathcal{X} \trianglelefteq z_k\right\}$$

is called the k-th-order attainment function of $\mathcal{X}$ (Grunert da Fonseca and Fonseca 2004).

Hence, MO performance can be *completely* described by the attainment function of order $k = m^*$. The higher the order of the attainment function, i.e., the more goals are considered with respect to their probability of being attained simultaneously, the more complete the performance description should be. However, *no* further gain of information will be obtained when the number k of goals considered exceeds m^*, because any higher-than-m^*-th-order attainment function could be theoretically derived from the attainment function of order m^*; see Task 1.1 in the proof of Theorem 1 in Grunert da Fonseca and Fonseca (2004).

When m^* is large, the *full* assessment of MO performance via the k-th-order attainment function with $k = m^*$ is rather impractical, although it remains of much theoretical interest. In fact, the information conveyed by the k-th-order attainment function is too rich for large k, as much from a computational as from a conceptual point of view. Since it is essentially a function of $d \times k$ variables, both computation and graphical visualization of the k-th-order attainment function are generally difficult.

For the purpose of optimizer comparison, which is the ultimate objective in any empirical study of optimizer performance, it is more advantageous to develop simpler, if only *partially* informative, performance criteria. This is also how statistical estimator performance is assessed in the context of the frequency school of inference (as has been discussed in Sect. 5.2.2).

5.5 Partial Aspects of Multiobjective Optimizer Performance

The *closeness* of the sampling distribution of statistical point estimators to the unknown estimand in $\mathbb{R}$, both in terms of *location* and *spread* (Sect. 5.2.2), is perhaps the most important aspect of estimator performance. Therefore, typical performance measures for point estimators include the mean-bias (*location*) and the variance or the standard deviation[3] (*spread*), with respect to the estimator sampling distribution. Other measures, such as the mean squared error, may also be considered, to account simultaneously for *location* and *spread*. For skewed sampling distributions, the preferred measures are often the median-bias for *location* and the inter-quartile range for *spread* (Hoaglin et al. 2000).

The mean and the variance of a univariate (sampling) distribution with support on $\mathbb{R}$ are, respectively, the *first-order moment* and the *second-order centered moment* of that distribution (Mood et al. 1974, p. 73). Accordingly, corresponding notions of moments in the context of random closed sets should lead to suitable partial performance measures for multiobjective optimizers (Sects. 5.5.1 and 5.5.2). Due to the complexity of the RNP set distributions, it is important to consider also other *higher-order moments*, as they are able to reflect some of the inter-point dependence structures within an RNP set (Sect. 5.5.3).

5.5.1 Distribution Location: The First-Order Attainment Function

It should be recognized that there is no *unique* definition of a mean (first-order moment) for a general random closed set. Instead, various mean definitions have been formulated depending on specific properties of the random sets, e.g. convexity, stationarity or compactness (Stoyan et al. 1995, Molchanov 2005). The first two classes of sets are not of interest in the context of multiobjective optimization, because neither the RNP set $\mathcal{X}$ nor its associated attained set $\mathcal{Y}$ are convex or stationary. A popular mean definition for compact random closed sets is the *Aumann-mean*, or selection expectation, which produces a convex set-valued mean (Molchanov 2005). However, even though the RNP set $\mathcal{X}$ is compact, it is generally *nonconvex*, as pointed out above. Hence, this mean definition is not suitable for MO performance description, either (Grunert da Fonseca et al. 2001).

The First-Order Attainment Function as a Covering Function

A function-valued mean definition which turns out to be applicable to the study of MOs is the so-called *covering function* or coverage function (Molchanov 2005, p. 23). Let $\mathbf{I}_{\{\cdot\}}(z) = \mathbf{I}\{z \in \cdot\}$ denote the indicator function defined over $z \in \mathbb{R}^d$. Then, a general random closed set $\mathcal{W} \subset \mathbb{R}^d$ with indicator function $\mathbf{I}_{\mathcal{W}}(z) =$

[3] The standard deviation of an estimator's sampling distribution is the estimator's *standard error*.

$\mathbf{I}\{z \in \mathcal{W}\}$ defines a binary random field $\{\mathbf{I}\{z \in \mathcal{W}\},\ z \in \mathbb{R}^d\}$, as described by Sivakumar and Goutsias (1996) and Goutsias (1998, p. 14). The *first-order moment* of such a binary random field is the expectation of the indicator function, that is,

$$\mathrm{E}\big[\,\mathbf{I}\{z \in \mathcal{W}\}\big] = \Pr\{z \in \mathcal{W}\}, \quad \text{for all } z \in \mathbb{R}^d \tag{5.1}$$

(Sivakumar and Goutsias 1996, p. 901). This expectation is known as the covering function of the random closed set $\mathcal{W}$ (Molchanov 2005, p. 176). It is interesting to note that the covering function may also be interpreted as the "membership function" of a fuzzy set (Nuñez-Garcia and Wolkenhauer 2002).

In the context of multiobjective optimization, it can now be easily concluded that the *first-order attainment function* of the corresponding RNP set $\mathcal{X}$ (Definition 5.4 with $k = 1$) is in fact the covering function of the attained set $\mathcal{Y}$, since

$$\begin{aligned}\Pr\{z \in \mathcal{Y}\} &= \Pr\big\{z \in \{y \in \mathbb{R}^d \mid X_1 \le y \ \vee\ X_2 \le y \ \vee \ldots \vee\ X_M \le y\}\big\}\\ &= \Pr\{X_1 \le z \ \vee\ X_2 \le z \ \vee \ldots \vee\ X_M \le z\}\\ &= \Pr\{\mathcal{X} \trianglelefteq z\} = \alpha_{\mathcal{X}}^{(1)}(z) = \alpha_{\mathcal{X}}(z).\end{aligned}$$

Hence, the first-order attainment function of the RNP set $\mathcal{X}$ is the first-order moment of the binary random field $\{\mathbf{I}\{z \in \mathcal{Y}\},\ z \in \mathbb{R}^d\}$ derived from $\mathcal{Y}$. For every goal $z \in \mathbb{R}^d$, it represents the expected value of the random *attainment indicator* $\mathbf{I}\{z \in \mathcal{Y}\} = \mathbf{I}\{\mathcal{X} \trianglelefteq z\}$. Thus, as a mean-like measure, $\alpha_{\mathcal{X}}(\cdot)$ can be used to describe the *location* of the MO outcome distribution which, in general, conveys *partial*, but relevant, information about the overall optimizer performance. Only when $M = 1 = m^*$ (with probability one), in which case the optimizer under study always produces one-point outcome sets $\mathcal{X} = \{X\}$, is the corresponding first-order attainment function identical to the *fully* characterizing multivariate cumulative distribution function $F_X(z) = \Pr\{X \le z\}$ (Grunert da Fonseca and Fonseca 2002).

The first-order attainment function has a very useful *interpretation* from an optimization point of view: for every goal z in objective space $\mathbb{R}^d$, it provides the probability of attaining z in a single optimization run. The larger this probability turns out to be over all $z \in \mathbb{R}^d$, the better the performance of the corresponding optimizer should be. In other words, to imply "good" optimizer performance on a given optimization problem, the first-order attainment function should show an early and steep increase in each dimension of the objective space, starting from its lower boundary of support, which should be identical to the true Pareto-optimal front[4]. Note that, in this sense, the attainment function can also provide some kind of *skewness* description for the optimizer outcome distribution.

When the true Pareto-optimal front $\mathcal{X}^*$ is known, a notion of *bias* may be constructed in terms of the difference between the first-order attainment function and the *ideal (first-order) attainment function* $\alpha_I(z) = \mathbf{I}\{\mathcal{X}^* \trianglelefteq z\}$. Note that $\alpha_I(\cdot)$

[4] If the optimizer is not technically able to reach (the whole of) the true Pareto-optimal front, the lower boundary of support of the first-order attainment function turns out to be (partially) greater than the true Pareto-optimal front. This situation may be termed "data contamination" (Grunert da Fonseca and Fieller 2006).

is zero below $\mathcal{X}^*$ and one otherwise. In this sense, the bias is a function of a goal $z \in \mathbb{R}^d$, and indicates how far from ideally the optimizer performs regarding the attainment of that goal, where "ideally" corresponds to a difference of zero (Grunert da Fonseca et al. 2001).

The First-Order Attainment Function and the Vorob'ev Median

The location of a random, *not* necessarily bounded, closed set $\mathcal{W} \subset \mathbb{R}^d$ may also be described by the *set-valued* Vorob'ev median (Molchanov 2005, p. 176ff) which is defined as

$$\mathrm{V}_{0.5}(\mathcal{W}) = \big\{z \in \mathbb{R}^d \mid \Pr\{z \in \mathcal{W}\} \geq 0.5\big\},$$

where $\Pr\{z \in \mathcal{W}\}$ is the covering function of $\mathcal{W}$ as given in (5.1). The Vorob'ev median of $\mathcal{W}$ is a closed, deterministic subset of $\mathbb{R}^d$ whose indicator function $\mathbf{I}_{\mathrm{V}_{0.5}(\mathcal{W})}(\cdot)$ approximates the covering function of $\mathcal{W}$ in the sense that it minimizes

$$\sup_{z \in \mathbb{R}^d} \big|\Pr\{z \in \mathcal{W}\} - \mathbf{I}_{\mathrm{V}_t(\mathcal{W})}(z)\big|$$

over all covering function *(upper) excursion sets* defined as

$$\mathrm{V}_t(\mathcal{W}) = \big\{z \in \mathbb{R}^d \mid \Pr\{z \in \mathcal{W}\} \geq t\big\}, \quad t \in (0, 1]. \tag{5.2}$$

A set $\mathrm{V}_t(\mathcal{W})$ is also called the *t-th (Vorob'ev) quantile* of $\mathcal{W}$, and its lower boundary is the level-t isoline of the covering function of $\mathcal{W}$ (Molchanov 2005, p. 175ff).

In the context of multiobjective optimization, where the covering function of the attained set $\mathcal{Y}$ is identical to the (first-order) attainment function of the RNP set $\mathcal{X}$, a level-t isoline of $\alpha_{\mathcal{X}}(\cdot)$ has been referred to as the "$t \times 100\%$-attainment surface" (Fonseca and Fleming 1996). An excursion set of $\alpha_{\mathcal{X}}(\cdot)$, as defined in (5.2), now results in the *t-th (Vorob'ev) quantile of the attained set* $\mathcal{Y}$. A particular way of describing the (central) *location* of an MO outcome distribution is therefore given by the definition of the *Vorob'ev median of the attained set* (Definition 5.5), which is a deterministic, closed, and unbounded set in objective space $\mathbb{R}^d$.

Definition 5.5. (Vorob'ev median of the attained set) The set in $\mathbb{R}^d$ denoted as

$$\mathrm{V}_{0.5}(\mathcal{Y}) = \big\{z \in \mathbb{R}^d \mid \alpha_{\mathcal{X}}(z) \geq 0.5\big\}$$

is the Vorob'ev median of the set of all goals $z \in \mathbb{R}^d$ attained by the RNP set $\mathcal{X}$.

At this stage, it is important to clarify that the so-called *Vorob'ev expectation* of a random closed set (Molchanov 2005, p. 177) is *not* suitable for the performance description of MOs, even though its definition is very similar to that of the Vorob'ev median. In fact, the definition of the Vorob'ev expectation of a set $\mathcal{W}$ is based on the minimization of

$$\left| \int_{\mathbb{R}^d} \Pr\{z \in \mathcal{W}\} - \mathbf{I}_{\mathrm{V}_t(\mathcal{W})}(z)\, dz \right|, \tag{5.3}$$

which only makes sense for finite integrals, as when the random closed set $\mathcal{W}$, as opposed to $\mathcal{Y}$, is compact. The problem may be overcome by considering integrals in (5.3) which are defined over some compact subset K of $\mathbb{R}^d$. Such a "restricted" Vorob'ev expectation could then be defined also for unbounded random closed sets, like the attained set $\mathcal{Y}$. Note that the Vorob'ev expectation is generally useless for random sets with zero volume, such as the random nondominated *point*-set $\mathcal{X}$.

Finally, given the knowledge of the true Pareto-optimal front $\mathcal{X}^*$, an upper excursion set of the (first-order) attainment function may be used to define some notion of bias with set character. For example, the difference

$$\{z \in \mathbb{R}^d \mid \mathcal{X}^* \trianglelefteq z\} - \mathrm{V}_{0.5}(\mathcal{Y})$$

between the set of all goals "attained" by $\mathcal{X}^*$ and the Vorob'ev median of $\mathcal{Y}$ may define such a bias-set. Its finite area within some compact reference set K could be associated with the level of optimizer performance.

5.5.2 Distribution Spread: The Variance Function

In contrast with the availability of several mean definitions, there are only a few discussions in the literature about how to formulate a "variance" of a random closed set. Furthermore, from the two location approaches presented in Sect. 5.5.1, only the first one seems to allow for a suitable variance extension. Consider again a general random closed set $\mathcal{W} \subset \mathbb{R}^d$ and the corresponding binary random field $\big\{\mathbf{I}\{z \in \mathcal{W}\},\ z \in \mathbb{R}^d\big\}$. Since the expression $\mathbf{I}\{z \in \mathcal{W}\}$ is a Bernoulli random variable with parameter value $p_z = \Pr\{z \in \mathcal{W}\}$ for all $z \in \mathbb{R}^d$, the *variance* of such a binary random field at any point $z \in \mathbb{R}^d$ may be given by $\Pr\{z \in \mathcal{W}\} \cdot (1 - \Pr\{z \in \mathcal{W}\})$.

Further, recall that in the context of multiobjective optimization the equality

$$\alpha_{\mathcal{X}}(z) = \Pr\{z \in \mathcal{Y}\},$$

holds for all $z \in \mathbb{R}^d$. Hence, the *spread* of an optimizer outcome distribution may be described by the *variance function* which, once again, is based on the first-order attainment function (Definition 5.6).

Definition 5.6. (Variance function) The function defined as $\mathrm{Var}_{\mathcal{X}} : \mathbb{R}^d \longrightarrow [0, 0.25]$ with

$$\mathrm{Var}_{\mathcal{X}}(z) = \alpha_{\mathcal{X}}(z) - [\alpha_{\mathcal{X}}(z)]^2$$

is called the variance function of $\mathcal{X}$ (Fonseca et al. 2005).

For every goal $z \in \mathbb{R}^d$, the variance function indicates the variability of the MO outcomes, across multiple runs, with respect to the expected attainment of that goal.

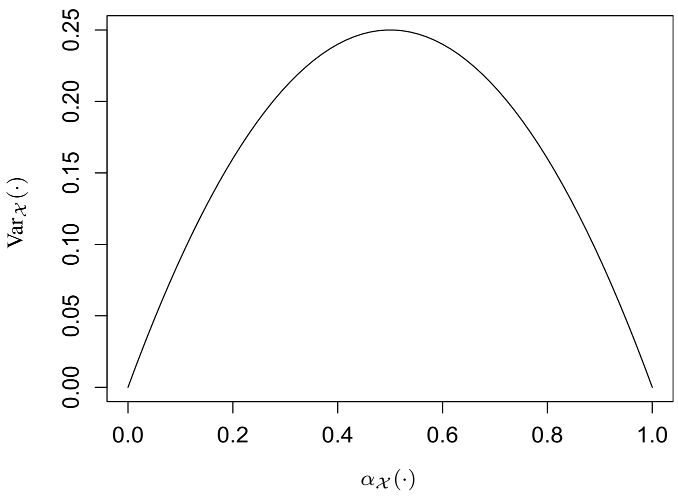


Fig. 5.2: First-order attainment function values versus variance function values

Smaller function values imply that, for the given point $z \in \mathrm{I\!R}^d$, 0-1 outcomes of the attainment indicator $\mathbf{I}\{z \in \mathcal{Y}\}$ vary less around $\alpha_{\mathcal{X}}(z)$. The variance function cannot show values beyond 0.25 as its maximum is reached for all $z \in \mathrm{I\!R}^d$ for which $\alpha_{\mathcal{X}}(z) = 0.5$. As illustrated in Fig. 5.2, the variance function is *fully* determined by the (first-order) attainment function. Hence, it does not convey any information about MO performance beyond that already transmitted by $\alpha_{\mathcal{X}}(\cdot)$.

The variance function, as defined, is a restricted version of the second-order centered moment of the binary random field $\{\mathbf{I}\{z \in \mathcal{Y}\},\ z \in \mathrm{I\!R}^d\}$, as explained below.

5.5.3 Inter-Point Dependence Structures: Second and Higher-Order Attainment Functions

Due to the complexity of the RNP set $\mathcal{X}$ (and the corresponding attained set $\mathcal{Y}$), it is generally not enough to describe its distribution in terms of *location* and *spread*. The random vectors $X_1, X_2, \ldots, X_M$ of the RNP set $\mathcal{X}$ (see Definition 5.2) are mutually *dependent* due to the imposed *nondominance* condition. As a consequence, the set distributions of $\mathcal{X}$ and $\mathcal{Y}$ are also determined by the kind and strength of the relationships within each of the possible pairs, triples, quadruples, *etc.*, of random vectors X_i in $\mathcal{X}$.

For example, the *second-order attainment function* $\alpha^{(2)}_{\mathcal{X}}(\cdot,\cdot)$ (see Definition 5.4 with $k = 2$) illustrates the second-order dependencies between all pairs of random vectors (X_i, X_j), $i \neq j$, originated in $\mathcal{X}$, since

$$\begin{aligned}
\alpha_{\mathcal{X}}^{(2)}(z_1, z_2) &= \Pr\Big\{\mathcal{X} \trianglelefteq z_1 \ \wedge\ \mathcal{X} \trianglelefteq z_2\Big\} \\
&= \Pr\Big\{(X_1 \leq z_1 \ \vee \ldots \vee\ X_M \leq z_1) \ \wedge\ (X_1 \leq z_2 \ \vee \ldots \vee\ X_M \leq z_2)\Big\} \\
&= \Pr\Big\{(X_1 \leq z_1 \ \wedge\ X_1 \leq z_2) \ \vee\ (X_1 \leq z_1 \ \wedge\ X_2 \leq z_2) \ \vee\ \ldots \\
&\qquad \ldots \vee\ (X_M \leq z_1 \ \wedge\ X_{M-1} \leq z_2) \ \vee\ (X_M \leq z_1 \ \wedge\ X_M \leq z_2)\Big\}.
\end{aligned}$$

Hence, the probability of simultaneously attaining two goals z_1 and z_2 in $\mathbb{R}^d$, as described by $\alpha_{\mathcal{X}}^{(2)}(z_1, z_2) = \alpha_{\mathcal{X}}^{(2)}(z_2, z_1)$, corresponds to the probability of these two goals being attained by at least one *pair* (X_i, X_j) of random vectors in $\mathcal{X}$. It is evident that the information of the first-order attainment function $\alpha_{\mathcal{X}}(z)$ is contained in that of the second-order attainment function when considering either $z_1 \leq z_2$ or $z_1 \geq z_2$.

The second-order attainment function is the *second-order moment* of the binary random field $\big\{\mathbf{I}\{z \in \mathcal{Y}\},\ z \in \mathbb{R}^d\big\}$ for points z_1 and z_2 in $\mathbb{R}^d$, as defined by Sivakumar and Goutsias (1996, p. 901). In combination with the first-order attainment function, the corresponding *second-order centered moment* may then be defined as the *covariance function* of $\mathcal{X}$.

Definition 5.7. (Covariance function) The function $\text{Cov}_{\mathcal{X}} : \mathbb{R}^{d\times 2} \longrightarrow [-0.25, 0.25]$ with

$$\text{Cov}_{\mathcal{X}}(z_1, z_2) = \alpha_{\mathcal{X}}^{(2)}(z_1, z_2) - \alpha_{\mathcal{X}}(z_1) \cdot \alpha_{\mathcal{X}}(z_2)$$

is called the covariance function of $\mathcal{X}$ (Fonseca et al. 2005).

The covariance function is of special interest for MO performance characterization, as it reflects the dependence between any two random attainment indicators $\mathbf{I}\{\mathcal{X} \trianglelefteq z_1\}$ and $\mathbf{I}\{\mathcal{X} \trianglelefteq z_2\}$. For instance, if the attainment of goal z_1 is independent from the attainment of goal z_2, then $\text{Cov}_{\mathcal{X}}(z_1, z_2)$ is zero. On the other hand, a positive covariance function value indicates a *positive* dependence between the two attainment indicators, in the sense that the attainment of goal z_1 tends to *coincide* with the attainment of goal z_2, while negative function values indicate that the attainment of goals z_1 and z_2 tends to be mutually exclusive.

Considering pairs of nondominated goals far apart from each other, negative covariance values may occur, for example, if the optimizer tends to converge to a different (small) region of the Pareto-optimal front in each run, instead of covering the whole front. On the other hand, positive covariance values between distant goals would be consistent with an optimizer which tends to approximate the whole front equally well, but to a different degree, in each run. In this case, some runs would be fully successful, whereas others would be consistently less successful across the whole front. These two situations suggest that the ideal case, where the MO outcome sets exhibit a "good uniform distribution" and a "large spread" along the true Pareto front (Zitzler 1999), may well correspond to intermediate covariance values, i.e., around zero. Note, however, that the covariance will always become nonnegative as one goal approaches the other, and will coincide with the variance when the two goals are equal.

The covariance function reaches its maximum value of 0.25 for all $z_1 = z_2 = z$ where $\alpha_{\mathcal{X}}(z) = 0.5$. The minimum value of -0.25 is possible for two goals which cannot be attained together, but where each one can be individually attained with probability 0.5. However, as a rule, the covariance function *cannot quantify* the strength of dependence between $\mathbf{I}\{\mathcal{X} \trianglelefteq z_1\}$ and $\mathbf{I}\{\mathcal{X} \trianglelefteq z_2\}$, since the possible range of its values for a pair of goals (z_1, z_2) depends on both $\alpha_{\mathcal{X}}(z_1)$ and $\alpha_{\mathcal{X}}(z_2)$. In other words, a perfect positive dependence between two attainment indicators might result in a covariance function value of 0.25, but, depending on $\alpha_{\mathcal{X}}(z_1)$ and $\alpha_{\mathcal{X}}(z_2)$, it could also result in some lower positive value. Unfortunately, the range of possible values of a (normalized) correlation function for such a pair of Bernoulli variables would still depend on $\alpha_{\mathcal{X}}(z_1)$ and $\alpha_{\mathcal{X}}(z_2)$, as noted by Hamrick (2009).

Higher-order dependencies between triples, quadruples, *etc.*, of random vectors X_i in $\mathcal{X}$ and, correspondingly, of random attainment indicators, can be investigated by the k-th-order attainment function, $k \geq 3$, which is the *k-th-order moment* of the binary random field $\left\{\mathbf{I}\{z \in \mathcal{Y}\},\ z \in \mathbb{R}^d\right\}$ for the points $z_1, \ldots, z_k$ in $\mathbb{R}^d$ (Sivakumar and Goutsias 1996, p. 901). Obviously, the information contained in $\alpha^{(k)}_{\mathcal{X}}(\cdot, \ldots, \cdot)$ covers the information given by all other lower-than-k-th-order attainment functions. If k is the maximum possible number of nondominated random vectors X_i in $\mathcal{X}$, then $\alpha^{(k)}_{\mathcal{X}}(\cdot, \ldots, \cdot)$ ultimately presents the *full* information about the optimizer outcome distribution, as already pointed out in Sect. 5.4.2.

5.6 Multiobjective Optimizer Performance Assessment: Estimation

So far, MO performance descriptions with the attainment-function approach have been discussed in this chapter solely from a *theoretical* point of view. For a given optimization problem and a chosen MO, however, attainment functions of any order k are entirely *unknown*, since no (model) assumptions have been made about the underlying optimizer outcome distribution (see Sect. 5.2.3). Therefore, the corresponding attainment functions should be estimated using a realized random sample of independent and identically distributed MO outcome sets, obtained through *multiple* optimization runs.

In the nonparametric context, the multivariate cumulative distribution function $F_X(z) = \Pr\{X \leq x\}$ of a random vector $X \in \mathbb{R}^d$ may be estimated via the multivariate empirical distribution function

$$F_n(z) = \frac{1}{n} \cdot \sum_{i=1}^{n} \mathbf{I}\{X_i \leq z\},$$

where $X_1, X_2, \ldots, X_n$ is a random sample of random vectors which are independent and identically distributed like the random vector X.

This simple idea of averaging over the n sample elements obtained can also be used to formulate a nonparametric *empirical estimator* for the first-order attainment function and, more generally, for the k-th-order attainment function:

Definition 5.8. (Empirical k-th-order attainment function) Let $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_n$ be a random sample of RNP sets which are independent and identically distributed like the RNP set $\mathcal{X} \subset \mathbb{R}^d$. Then, the discrete function defined as $\alpha_n^{(k)} : \mathbb{R}^{d\times k} \longrightarrow [0,1]$ with

$$\begin{aligned} \alpha_n^{(k)}(z_1, \ldots, z_k) &= \alpha_n^{(k)}(\mathcal{X}_1, \ldots, \mathcal{X}_n; z_1, \ldots, z_k) \\ &= \frac{1}{n} \sum_{i=1}^{n} \mathbf{I}\{\mathcal{X}_i \trianglelefteq z_1 \ \wedge \ \ldots \ \wedge \ \mathcal{X}_i \trianglelefteq z_k\} \\ &= \frac{1}{n} \sum_{i=1}^{n} \mathbf{I}\{\mathcal{X}_i \trianglelefteq z_1\} \cdot \ \ldots \ \cdot \ \mathbf{I}\{\mathcal{X}_i \trianglelefteq z_k\}, \end{aligned}$$

is called the empirical k-th-order attainment function of $\mathcal{X}$.

The idea of the *empirical first-order attainment function* of $\mathcal{X}$ can be traced back to the early paper of Fonseca and Fleming (1996), while its mathematical formulation

$$\begin{aligned} \alpha_n(z) &= \alpha_n(\mathcal{X}_1, \ldots, \mathcal{X}_n; z) \\ &= \frac{1}{n} \sum_{i=1}^{n} \mathbf{I}\{\mathcal{X}_i \trianglelefteq z\}, \end{aligned}$$

was proposed by Grunert da Fonseca et al. (2001). The fact that the above empirical functions are defined as finite sums of indicator functions easily explains their *discrete* nature. For increasing sample size n (number of optimization runs), it is expected that these step-functions converge to their unknown theoretical counterpart, like the multivariate empirical distribution function $F_n(\cdot)$ does.

The visualization of such discrete functions turns out to be a challenge in all but the simplest cases, where $k = 1$ and $d \leq 2$. Clearly, the single-objective case ($d = 1$) corresponds to the visualization of a univariate empirical cumulative distribution function, which offers no difficulty. In the biobjective case ($d = 2$), the empirical first-order attainment function $\alpha_n(\cdot)$ can be visualized quite effectively by determining its isolines at various levels $t \in (0,1)$ and plotting them on the objective plane, as illustrated in Fig. 5.3 (see also Chapter 9).

Finally, for the case $k = 2$ and $d = 2$, a projection technique has been suggested in Fonseca et al. (2005), which consists of fixing one goal z^* and depicting the isolines of the marginal function $\alpha_n^{(2)}(\cdot, z^*) = \alpha_n^{(2)}(z^*, \cdot)$ at several levels t, as before. Obviously, there are infinitely many possible positions for z^* in $\mathbb{R}^d$, and one can only hope that $\alpha_n^{(2)}(z^*, \cdot)$ exhibits similar behavior in the vicinity of the current

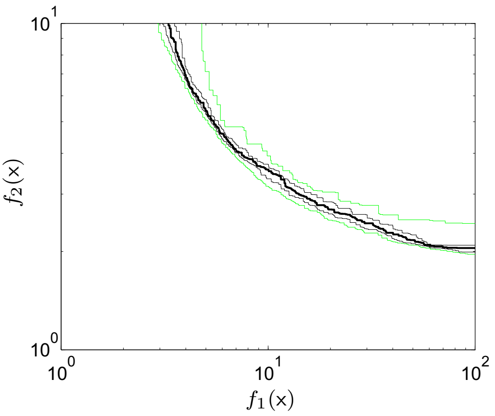


Fig. 5.3: Graphical representation of an empirical first-order attainment function based on $n = 21$ biobjective optimization runs, with isolines at levels ϵ, 0.25, 0.5, 0.75, and $1 - \epsilon$, where ϵ is some arbitrarily small positive value, and $f_1(x)$ and $f_2(x)$ denote the two objective functions to be minimized (from Fonseca and Fleming 1996)

choice of z^* in objective space. A better perception of $\alpha_n^{(2)}(\cdot,\cdot)$ may be acquired through an interactive session, where the position of the goal z^* is continuously changed.

The intrinsic importance of the empirical k-th-order attainment function lies, nevertheless, in its potential to provide a tool for the performance *comparison* of two or more MOs, as discussed in the next section.

5.7 Multiobjective Optimizer Performance Comparison: Hypothesis Testing

Ultimately, *assessing* the performance of estimators, each on its own, is not sufficient. In order to choose among competing alternative estimators for the same unknown estimand, one needs to *compare* them with each other with respect to their performance. The preferred estimator is then the one that shows, for example, minimum (zero) bias and uniform minimum variance over all potential values of the estimand (Lehmann and Casella 1998, p. 85).

Most point estimators have the potential to either underestimate or overestimate the unknown estimand, the actual value of which must be considered when determining the bias. MOs, on the other hand, can only *over*estimate the unknown, true Pareto-optimal front (in a minimization context). Therefore, the corresponding first-order moments can be compared *directly* without reference to the true front, and explicit knowledge of the bias is not required.

Nevertheless, comparing MO performance is not easier than comparing the performance of ordinary point estimators! Whereas most properties of statistical point estimators are expressed in terms of single numbers (e.g. the bias, the variance, *etc.*), and can be easily compared, the first and higher-order moments associated with the attained set $\mathcal{Y}$ are unknown functions defined over $\mathbb{R}^{d\times k}$, $d > 1$, $k \geq 1$.

5.7.1 Two-Sided Test Problem

The simplest kind of performance comparison between two optimizers is concerned with whether or not they perform differently. In statistical practice, this leads to the formulation of a *two-sided*, *two-sample* test problem involving the two MOs, A and B, which are executed independently from each other. Taking the k-th-order attainment function, the corresponding test problem may be stated as:

$$H_0 : \ \alpha^{(k)}_{\mathcal{X}_A}(z_1,\dots,z_k) = \alpha^{(k)}_{\mathcal{X}_B}(z_1,\dots,z_k) \quad \text{for all } (z_1,\dots,z_k) \in \mathbb{R}^{d\times k}$$

versus

$$H_1 : \ \alpha^{(k)}_{\mathcal{X}_A}(z_1,\dots,z_k) \neq \alpha^{(k)}_{\mathcal{X}_B}(z_1,\dots,z_k) \quad \text{for at least one } (z_1,\dots,z_k) \in \mathbb{R}^{d\times k}, \tag{5.4}$$

where $\mathcal{X}_A$ and $\mathcal{X}_B$ denote the (random) outcome sets of optimizers A and B, respectively (see Fonseca et al. 2005, for the cases $k = 1$ and $k = 2$).

In general, the null hypothesis H_0 does *not* imply that the entire set distributions of $\mathcal{X}_A$ and of $\mathcal{X}_B$ are the same. This is only true when both $\alpha^{(k)}_{\mathcal{X}_A}(\cdot,\dots,\cdot)$ and $\alpha^{(k)}_{\mathcal{X}_B}(\cdot,\dots,\cdot)$ can fully characterize the distributions of $\mathcal{X}_A$ and of $\mathcal{X}_B$, that is, when k is (at least) the maximum possible number of nondominated solutions per outcome of A and of B. Still, the two optimizers A and B may be said to be *equivalent*[5] in performance under H_0 with respect to the k-th-order attainment function considered.

Such a definition of equivalence for the two MOs becomes more restrictive with larger k, since

$$H_0^{(1)} \supset H_0^{(2)} \supset \dots \supset H_0^{(k)}, \tag{5.5}$$

where $H_0^{(i)}$ is the null hypothesis of the test problem based on the i-th-order attainment function. In other words, the size of H_0 in test problem (5.4) decreases as the order of the attainment function increases, and the following implications hold for $i = 1,\dots,k-1$:

$$H_0^{(i+1)} \text{ is true} \quad \Longrightarrow \quad H_0^{(i)}, H_0^{(i-1)}, \dots, H_0^{(1)} \text{ are true.} \tag{5.6}$$

$$H_0^{(i)} \text{ is not true} \quad \Longrightarrow \quad H_0^{(i+1)}, H_0^{(i+2)}, \dots, H_0^{(k)} \text{ are not true.} \tag{5.7}$$

[5] By analogy to the so-called "equivalence tests" commonly applied in environmental statistics, see for example Manly (2001, p. 184f).

The application of a statistical hypothesis test to the general problem H_0 v.s. H_1 for a given significance level $\alpha \in (0,1)$, allows one of two decisions to be made: "reject H_0" or "do not reject H_0". In contrast to a merely visual comparison of the two corresponding empirical attainment functions (if this is, at all, possible), the advantage of this approach is that any rejection of H_0 would imply *statistically significant evidence* of a difference in performance between the two MOs A and B. In other words, the probability of committing a Type I error, i.e., concluding that the performances of A and B are not equivalent when they are, in fact, equivalent, is controlled to be less than or equal to the significance level α.

5.7.2 Permutation Test Procedure

The Kolmogorov-Smirnov (KS) two-sample test (Conover 1999, p. 456ff) is a well-known hypothesis test for the comparison of two unknown univariate cumulative distribution functions F_1 and F_2. Its test statistic $D_{n,m}$ is based on the maximum (vertical) absolute difference between the corresponding empirical distribution functions, F_n^1 and F_m^2, from two independently drawn random samples of size n and m, respectively. According to the KS-decision rule, the null hypothesis H_0 (claiming the equality of F_1 and F_2) is rejected for a given significance level α, if

$$D_{n,m} = \sup_{z \in \mathbb{R}} \left| F_n^1(z) - F_m^2(z) \right| > d_{n;m;1-\alpha},$$

where the critical value $d_{n;m;1-\alpha}$ is the $(1-\alpha)$-quantile of the sampling distribution of $D_{n,m}$ under H_0. The univariate two-sample KS test is "distribution-free," which means that this sampling distribution (under H_0) is invariant with respect to the underlying population distribution, and its quantiles may be determined and tabulated for different values of n and m. In addition, a *p-value*, defined as the smallest significance level for which H_0 could still be rejected for the given data, can be calculated (Conover 1999, p. 101, 458). In other words, reject H_0 if $p \leq \alpha$.

Attainment functions of any order k can be seen as generalizations of the (univariate) cumulative distribution function, so that the KS two-sample test motivates the formulation of KS-*like* two-sample tests for the comparison of two attainment functions (Fonseca et al. 2005). Thus, for the test problem given in (5.4) and significance level α, reject H_0 if

$$D_{n,m}^{(k)} = \sup_{(z_1,\ldots,z_k) \in \mathbb{R}^{d \times k}} \left| \alpha_n^A(z_1,\ldots,z_k) - \alpha_m^B(z_1,\ldots,z_k) \right| > d_{n;m;1-\alpha}^{(k)}, \quad (5.8)$$

where $\alpha_n^A(\cdot,\ldots,\cdot)$ and $\alpha_m^B(\cdot,\ldots,\cdot)$ are, respectively, the empirical k-th-order attainment functions determined from n runs of optimizer A and from m runs of optimizer B. Note that, for simplicity, the upper index "(k)" has been removed from the notation of these functions.

In contrast to the univariate two-sample KS test, and similarly to the multivariate two-sample KS test, which is not distribution free (Bickel 1969), the critical value $d^{(k)}_{n;m;1-\alpha}$ in (5.8) is no longer invariant for given α, k, n and m. This problem may be side-stepped by conditioning on the given sample data and using the so-called "permutation argument" (Efron and Tibshirani 1993, Good 2000) to avoid referring to the *unknown* population distribution(s) under the null hypothesis.

The permutation argument states that all possible orderings of the elements in a sample are equally likely if the sample elements are *exchangeable*, which happens, for example, if they are independently drawn from the same distribution. Strictly speaking, this exchangeability condition is satisfied only when the two MOs have identical outcome distributions under H_0, i.e., when H_0 is simple, but it is often acceptable to relax this condition to accommodate suitable composite null hypotheses (Efron and Tibshirani 1993, p. 217) at the expense of the test becoming approximate rather than exact. Note that $H_0^{(k)}$ approaches the simple null hypothesis as k increases.

Given two samples of outcome sets generated, respectively, by n and m runs of optimizers A and B, the sampling distribution of the test statistic $D^{(k)}_{n,m}$ under $H_0^{(k)}$ may then be replaced by its "permutation distribution," which can be determined as follows:

1. Enumerate all possible permutations of the *pooled* sample data.
2. For each permutation, associate the first n outcome sets with optimizer A and the remaining m outcome sets with optimizer B.
3. Compute the test statistic for each permutation.

The required critical value is the $(1-\alpha)$-quantile of the frequency distribution of the test-statistic values thus obtained.

In practice, permutation distributions are usually approximated using a (large) number of permutations generated at random with equal probability, except when n and m are sufficiently small for exhaustive enumeration to be feasible. Such a randomization approach has been implemented for the two-sided KS-like tests based on the first and second-order attainment functions, and illustrative results from the comparison of two MOs in a simulation study are presented in Fonseca et al. (2005).

5.7.3 Multistage Testing

The above permutation test procedure has been presented for arbitrary values of k, and it may not be clear which value of k should be used in a particular, concrete comparative study. Recall that, by selecting some value $k = k^*$, one is establishing a notion of *equivalence in performance* between two MOs, based on the equality of the corresponding k^*-th-order attainment functions, and that, according to (5.5), the higher the value of k^*, the stricter this notion of equivalence becomes. Therefore, one needs to agree upon an order k^* such that any practically-relevant performance differences between the two MOs may be detected through the rejection of $H_0^{(k^*)}$.

As an example, if $k^* = 1$, only differences in location will be detectable, whereas a large k^* value, such as $k^* = 5$, may lead to the detection of performance differences difficult to interpret from an optimization point of view, even if statistically significant. In fact, very high-order moments are seldom considered in statistical practice.

Depending on the sample sizes n and m, and on the size of the outcome sets obtained, the determination of critical values for the hypothesis tests based on higher-order attainment functions can be computationally very demanding, even for $k = 2$ (Fonseca et al. 2005). Although the ever increasing availability of computing power should contribute to alleviate this difficulty, a multistage approach, allowing lower-order hypotheses to be tested before higher-order ones, would have the additional benefit of detecting more fundamental performance differences, such as differences in location, before more subtle ones, whenever possible.

Having selected k^* and a global significance level α, such a multistage testing procedure may begin by applying the test based on the *first*-order attainment function (Test 1). If $H_0^{(1)}$ can be rejected at the individual significance level $\alpha^* = \alpha/k^*$, statistically-significant evidence of a difference in performance between the two optimizers with respect to the k^*-th-order attainment function has been found. If $H_0^{(1)}$ cannot be rejected and $k^* = 1$, no such statistical evidence could be found. If $H_0^{(1)}$ cannot be rejected and $k^* > 1$, testing should continue for increasing values of k and the same significance level α^* until, either $H_0^{(k)}$ is rejected for some order $k < k^*$, or the result of the test based on the k^*-th-order attainment function (Test k^*) is obtained, as illustrated in Fig. 5.4.

The above multistage methodology is justified by the implications given in (5.6) and (5.7). In order to guarantee that the *equivalence in performance* null hypothesis of order k^* is incorrectly rejected with a probability *not* exceeding the given global significance level α, the significance levels of the individual tests have been adjusted by the well-known simple Bonferroni method (Shaffer 1995, p. 569). As a result, the global $\Pr\{\text{type I error}\}$ is bounded above by the prespecified global significance level α, as required. Formally:

$$
\begin{aligned}
&\Pr\{\text{type I error}\} \\
&= \Pr\Big\{\text{reject } H_0^{(1)} \;\vee\; (\text{not reject } H_0^{(1)} \;\wedge\; \text{reject } H_0^{(2)}) \\
&\qquad \vee (\text{not reject } H_0^{(1)} \;\wedge\; \text{not reject } H_0^{(2)} \;\wedge\; \text{reject } H_0^{(3)}) \\
&\qquad \ldots \\
&\qquad \vee (\text{not reject } H_0^{(1)} \;\wedge\; \ldots \;\wedge\; \text{not reject } H_0^{(k^*-1)} \;\wedge\; \text{reject } H_0^{(k^*)}) \;\Big|\; H_0^{(k^*)} \text{ true}\Big\} \\
&= \Pr\Big\{\text{reject } H_0^{(1)} \;\vee\; \text{reject } H_0^{(2)} \;\vee\; \ldots \;\vee\; \text{reject } H_0^{(k^*)} \;\Big|\; H_0^{(k^*)} \text{ true}\Big\} \\
&\approx \Pr\Big\{\text{reject } H_0^{(1)} \;\vee\; \text{reject } H_0^{(2)} \;\vee\; \ldots \;\vee\; \text{reject } H_0^{(k^*)} \;\Big|\; C_{per}\Big\} \\
&\leq \sum_{i=1}^{k^*} \Pr\Big\{\text{reject } H_0^{(i)} \;\Big|\; C_{per}\Big\} \leq \sum_{i=1}^{k^*} \alpha^* = \alpha,
\end{aligned}
$$

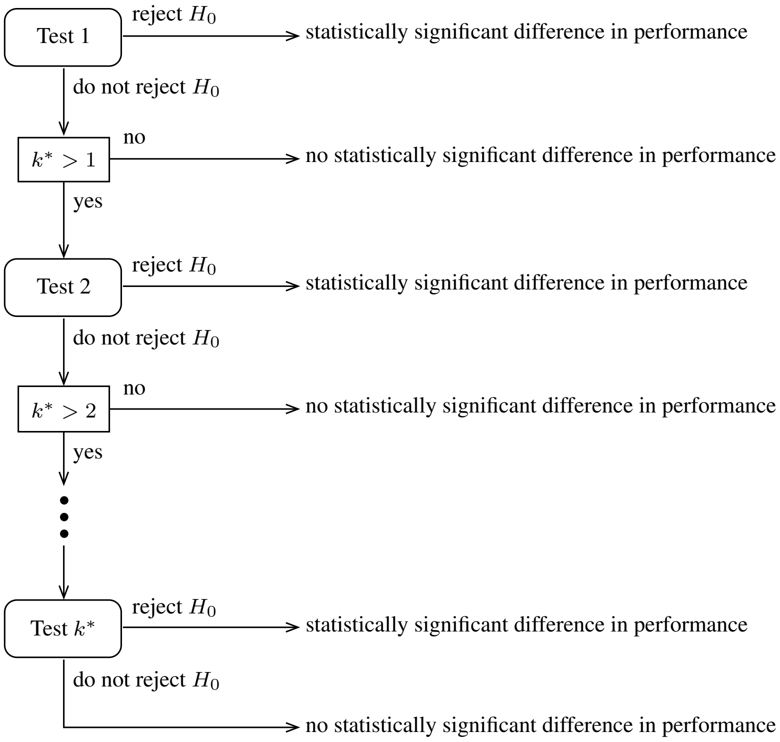

Test 1
reject H_0
statistically significant difference in performance
do not reject H_0
$k^* > 1$
no
no statistically significant difference in performance
yes
Test 2
reject H_0
statistically significant difference in performance
do not reject H_0
$k^* > 2$
no
no statistically significant difference in performance
yes
Test k^*
reject H_0
statistically significant difference in performance
do not reject H_0
no statistically significant difference in performance

Fig. 5.4: Multistage testing procedure. Individual test decisions are made at significance level $\alpha^* = \alpha/k^*$

where C_{per} denotes the collection of random permutations generated from the observed outcome sets of optimizers A and B.

In addition to being very simple, the Bonferroni method is also known to be quite conservative, leading to an unnecessary tendency not to reject the null hypothesis. More elaborate and less conservative adjustments should be possible, and are the subject of future work.

5.7.4 One-Sided Tests

When comparing a newly-developed MO (optimizer A) with an existing one (optimizer B), it may be more interesting to ask whether, at least in some sense, the new optimizer exhibits better performance than the existing one (the "control"). Still considering minimization, this leads to the formulation of the one-sided test problem:

$$H_0 : \alpha^{(k)}_{\mathcal{X}_A}(z_1,\ldots,z_k) \le \alpha^{(k)}_{\mathcal{X}_B}(z_1,\ldots,z_k) \quad \text{for all } (z_1,\ldots,z_k) \in \mathbb{R}^{d\times k}$$

versus

$$H_1 : \alpha^{(k)}_{\mathcal{X}_A}(z_1,\ldots,z_k) > \alpha^{(k)}_{\mathcal{X}_B}(z_1,\ldots,z_k) \quad \text{for at least one } (z_1,\ldots,z_k) \in \mathbb{R}^{d\times k},$$

with the test statistic:

$$\sup_{(z_1,\ldots,z_k)\in\mathbb{R}^{d\times k}} \left[\alpha^A_n(z_1,\ldots,z_k) - \alpha^B_m(z_1,\ldots,z_k)\right].$$

Critical values, in terms of $(1-\alpha)$-quantiles, as well as p-values, can be determined from the permutation distribution of this test statistic, which can be generated in exactly the same way as for the two-sided test. In addition, since the implications (5.6) and (5.7) also hold for the one-sided null hypotheses, the multistage testing procedure described earlier may also be legitimately applied.

5.8 Discussion and Future Perspectives

The attainment-function approach has been motivated by ideas from classical statistical inference theory (Sect. 5.2). It acknowledges that the solution sets produced by stochastic multiobjective optimizers are *random* entities which, in objective space, correspond to what can be defined as random nondominated point sets (Sect. 5.3). As a consequence, optimizer performance is studied through a description of the *probability distribution* of those random sets, both theoretically, with the support of probability theory (Sect. 5.4 and Sect. 5.5), and empirically, by means of statistical estimation and hypothesis testing using the results of multiple and independent optimization runs (Sect. 5.6 and Sect. 5.7). In all of these considerations, a central role has been given to the notion of *attainment function*, which can be seen as a hierarchy of different functions of increasing order. While the first-order attainment function provides partially-informative performance descriptions with respect to *location* and *spread* of the optimizer outcome distribution, second and higher-order versions of the attainment function are able to address the *inter-point dependence structures* of the optimization outcomes up to a full description of their distribution.

Despite its theoretical foundations, and although it continues to attract some level of interest from the scientific community, the attainment-function approach is still far from seeing widespread adoption in the experimental evaluation of multiobjective optimizer performance. Unary quality indicators, which describe the quality of given solution sets in terms of single values, and, eventually, binary quality indicators, which describe the difference in quality between two solution sets also as a single value, are currently much more popular (see Zitzler et al. 2003, for a review). The reasons for this would seem to range from the fact that single values, possibly averaged over a number of runs, are generally easier to interpret than functions defined over the whole objective space, to the fact that the computation of the empirical attainment function for more than two objectives has remained a challenge, despite continued efforts in that direction. In contrast, much more is known about the computation of the hypervolume indicator, for example (Bringmann and Friedrich 2008, Beume et al. 2009).

One important link between the two approaches can be established by relating the full distribution of indicator values for a given optimizer on a given problem to a corresponding attainment function of sufficiently high order. Preliminary results suggest that this should indeed be possible, at least for some quality indicators known to possess good properties. This will also contribute to a better understanding of what aspects of the distribution of RNP sets quality indicators really measure, and to what extent different indicators may complement each other.

Finally, an in-depth analysis of the statistical properties of the empirical k-th-order attainment function as an estimator, and of the hypothesis tests based on it, is still needed, as are generalizations of the methodology to encompass more than two optimizers and/or multiple problem instances. The development of suitable parametric or semiparametric models of MO outcome distributions, even if they only correspond to some ideal combination of problem and optimizer, is another possible direction for future research.

References

Ben-Tal A (1980) Characterization of Pareto and lexicographic optimal solutions. In: Fandel G, Gal T (eds) Multiple Criteria Decision Making. Theory and Applications, Lecture Notes in Economics and Mathematical Systems, vol 177, Springer, pp 1–11

Beume N, Fonseca CM, López-Ibáñez M, Paquete L, Vahrenhold J (2009) On the complexity of computing the hypervolume indicator. IEEE Transactions on Evolutionary Computation 13(5):1075–1082

Bickel PJ (1969) A distribution free version of the Smirnov two sample test in the p-variate case. The Annals of Mathematical Statistics 40(1):1–23

Bringmann K, Friedrich T (2008) Approximating the volume of unions and intersections of high-dimensional geometric objects. In: Hong SH, Nagamochi H, Fukunaga T (eds) Algorithms and Computation, 19th International Symposium, ISAAC 2008, Gold Coast, Australia, December 2008, Proceedings, Lecture Notes in Computer Science, vol 5369, Springer, pp 436–447

Conover WJ (1999) Practical Nonparametric Statistics, 3rd edn. Wiley Series in Probability and Statistics, Wiley, New York, NY

Daley DJ, Vere-Jones D (1988) An Introduction to the Theory of Point Processes. Springer Series in Statistics, Springer

Efron B, Tibshirani RJ (1993) An Introduction to the Bootstrap. No. 57 in Monographs on Statistics and Applied Probability, Chapman and Hall, New York, NY

Fonseca CM, Fleming PJ (1996) On the performance assessment and comparison of stochastic multiobjective optimizers. In: Voigt HM, Ebeling W, Rechenberg I, Schwefel HP (eds) Parallel Problem Solving from Nature – PPSN IV. International Conference on Evolutionary Computation. The 4th Conference on Parallel Problem Solving from Nature, Berlin, Germany, September 1996, Proceedings, Lecture Notes in Computer Science, vol 1141, Springer, pp 584–593

Fonseca CM, Grunert da Fonseca V, Paquete LF (2005) Exploring the performance of stochastic multiobjective optimisers with the second-order attainment function. In: Coello Coello CA, Aguirre AH, Zitzler E (eds) Evolutionary Multi-Criterion Optimization. Third International Conference, EMO 2005, Guanajuato, Mexico, March 2005, Proceedings, Lecture Notes in Computer Science, vol 3410, Springer, pp 250–264

Gilks WR, Richardson S, Spiegelhalter DJ (1996) Introducing Markov chain Monte Carlo. In: Gilks WR, Richardson S, Spiegelhalter DJ (eds) Markov Chain Monte Carlo in Practice, Chapman and Hall, London, chap 1, pp 1–19

Good P (2000) Permutation Tests: A Practical Guide to Resampling Methods for Testing Hypotheses, 2nd edn. Springer Series in Statistics, Springer

Goutsias J (1998) Modeling random shapes: An introduction to random closed set theory. Technical Report JHU/ECE 90-12, Department of Electrical and Computer Engineering, Image Analysis and Communications Laboratory, The Johns Hopkins University, Baltimore, MD 21218

Grunert da Fonseca V, Fieller NRJ (2006) Distortion in statistical inference: The distintion between data contamination and model deviation. Metrika 63:169–190

Grunert da Fonseca V, Fonseca CM (2002) A link between the multivariate cumulative distribution function and the hitting function for random closed sets. Statistics & Probability Letters 57(2):179–182

Grunert da Fonseca V, Fonseca CM (2004) A characterization of the outcomes of stochastic multiobjective optimizers through a reduction of the hitting function test sets. Technical report, Centro de Sistemas Inteligentes, Faculdade de Ciências e Tecnologia, Universidade do Algarve

Grunert da Fonseca V, Fonseca CM, Hall AO (2001) Inferential performance assessment of stochastic optimisers and the attainment function. In: Zitzler E, Deb K, Thiele L, Coello Coello CA, Corne D (eds) Evolutionary Multi-Criterion Optimization. First International Conference, EMO 2001, Zurich, Switzerland, March 2001, Proceedings, Lecture Notes in Computer Science, vol 1993, Springer, pp 213–225

Hamrick J (2009) Linear dependence between two Bernoulli random variables. http://demonstrations.wolfram.com/LinearDependenceBetweenTwoBernoulli RandomVariables/, accessed on 3/05/2009

Harding EF, Kendall DG (eds) (1974) Stochastic Geometry: A Tribute to the Memory of Rollo Davidson. Series in Probability and Mathematical Statistics, Wiley, New York, NY

Hoaglin DC, Mosteller F, Tukey JW (eds) (2000) Understanding Robust and Exploratory Data Analysis. Wiley Classics Library, Wiley, New York, NY

Hüsler J, Cruz P, Hall A, Fonseca CM (2003) On optimization and extreme value theory. Methodology and Computing in Applied Probability 5(2):183–195

Lehmann EL, Casella G (1998) Theory of Point Estimation, 2nd edn. Springer Texts in Statistics, Springer

Manly BFJ (2001) Statistics for Environmental Science and Management. Chapman and Hall, London

Matheron G (1975) Random Sets and Integral Geometry. Wiley, New York, NY

Molchanov I (2005) Theory of Random Sets. Probability and Its Applications, Springer

Mood AM, Graybill FA, Boes DC (1974) Introduction to the Theory of Statistics, 3rd edn. McGraw-Hill Series in Probability and Statistics, McGraw-Hill Book Company, Singapore

Nuñez-Garcia J, Wolkenhauer O (2002) Random set system identification. IEEE Transactions on Fuzzy Systems 10(3):287–296

Shaffer JP (1995) Multiple hypothesis testing. Annual Review of Psychology 46:561–584

Sivakumar K, Goutsias J (1996) Binary random fields, random closed sets, and morphological sampling. IEEE Transactions on Image Processing 5(6):899–912

Stoyan D, Kendall WS, Mecke J (1995) Stochastic Geometry and its Applications, 2nd edn. Wiley Series in Probability and Statistics, Wiley, New York, NY

Taylor P (1999) Practical Foundations of Mathematics. Cambridge Studies in Advanced Mathematics, Cambridge University Press, Cambridge, UK

Zitzler E (1999) Evolutionary algorithms for multiobjective optimization: Methods and applications. PhD thesis, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology, Zurich

Zitzler E, Thiele L, Laumanns M, Fonseca CM, Grunert da Fonseca V (2003) Performance assessment of multiobjective optimizers: An analysis and review. IEEE Transactions on Evolutionary Computation 7(2):117–132

Chapter 6
Algorithm Engineering: Concepts and Practice

Markus Chimani and Karsten Klein

Abstract Over the last years the term *algorithm engineering* has become wide spread synonym for experimental evaluation in the context of algorithm development. Yet it implies even more. We discuss the major weaknesses of traditional "pen and paper" algorithmics and the ever-growing gap between theory and practice in the context of modern computer hardware and real-world problem instances. We present the key ideas and concepts of the central *algorithm engineering cycle* that is based on a full feedback loop: It starts with the design of the algorithm, followed by the analysis, implementation, and experimental evaluation. The results of the latter can then be reused for modifications to the algorithmic design, stronger or input-specific theoretic performance guarantees, etc. We describe the individual steps of the cycle, explaining the rationale behind them and giving examples of how to conduct these steps thoughtfully. Thereby we give an introduction to current algorithmic key issues like I/O-efficient or parallel algorithms, succinct data structures, hardware-aware implementations, and others. We conclude with two especially insightful success stories—shortest path problems and text search—where the application of algorithm engineering techniques led to tremendous performance improvements compared with previous state-of-the-art approaches.

6.1 Why Algorithm Engineering?

"Efforts must be made to ensure that promising algorithms discovered by the theory community are implemented, tested and refined to the point where they can be usefully applied in practice. [...] to increase the impact of theory on key application areas."
[Aho et al. (1997), *Emerging Opportunities for Theoretical Computer Science*]

Markus Chimani · Karsten Klein
Algorithm Engineering, TU Dortmund, Germany e-mail: {markus.chimani|karsten.klein}@tu-dortmund.de

For a long time in classical algorithmics, the analysis of algorithms for combinatorial problems focused on the theoretical analysis of asymptotic worst-case runtimes. As a consequence, the development of asymptotically faster algorithms—or the improvement of existing ones—was a major aim. Sophisticated algorithms and data structures have been developed and new theoretical results were achieved for many problems.

However, asymptotically fast algorithms need not be efficient in practice: The asymptotic analysis may hide huge constants, the algorithm may perform poorly on typical real-world instances, where alternative methods may be much faster, or the algorithm may be too complex to be implemented for a specific task. Furthermore, modern computer hardware differs significantly from the *von Neumann* model (1993 republication) that often forms the basis of theoretical analyses. For use in real-world applications we therefore need robust algorithms that do not only offer good asymptotic performance but are also designed and experimentally evaluated to meet practical demands.

The research field of *algorithm engineering* copes with these problems and intends to bridge the gap between the efficient algorithms developed in algorithmic theory and the algorithms used by practitioners. In the best case, this may lead to algorithms that are asymptotically optimal and at the same time have excellent practical behavior.

An important goal of algorithm engineering is also to speed up the transfer of algorithmic knowledge into applications. This may be achieved by developing algorithms and data structures that have competitive performance but are still simple enough to be understood and implemented by practitioners. Additionally, free availability of such implementations in well-documented algorithm libraries can foster the use of state-of-the-art methods in real-world applications.

6.1.1 Early Days and the Pen-and-Paper Era

In the early days of computer algorithms, during the 1950s and 1960s, many pioneers also provided corresponding code for new algorithms. The classic and timeless pioneering work by Donald Knuth (1997), first published in 1968, gives a systematic approach to the analysis of algorithms and covers the aspect of algorithm implementation issues in detail without the restriction to a specific high-level language. Knuth once condensed the problems that arise due to the gap between theory and practice in the famous phrase

> "Beware of bugs in the above code; I have only proved it correct, not tried it."
> [D. E. Knuth]

During the following two decades, which are often referred to as the "pen-and-paper era" of algorithmics, the focus shifted more towards abstract high-level description of new algorithms. The algorithmic field saw many advances regarding new and improved algorithms and sophisticated underlying data structures. However, implementation issues were only rarely discussed; no implementations were provided or evaluated.

This led to a situation where on the one hand algorithms that may have been successful in practice were not published, as they did not improve on existing ones in terms of asymptotic runtime while on the other hand, the theoretically best algorithms were not used in practice as they were too complex to be implemented.

6.1.2 Errors

This situation had another side-effect: When algorithms are theoretically complex and never get implemented, errors in their design may remain undetected. However, the appearance of errors is inevitable in scientific research, as stated in the well-known quote

"If you don't make mistakes, you're not working on hard enough problems." [F. Wikzek]

Indeed are were a number of prominent examples: In 1973, Hopcroft and Tarjan (1973) presented the first linear-time algorithm to decompose a graph into its triconnected components, an algorithmic step crucial for graph-theoretic problems that build upon it. However, their description was flawed and it was not until 2001 (Gutwenger and Mutzel 2001)—when the highly complex algorithm was first implemented—that this was detected and fixed.

Another prominent example that we will revisit later in a different context is *planarity testing*, i.e., given a graph, we ask if it can be drawn in the plane without any crossings. Already the first algorithm (Auslander and Parter 1961) (requiring cubic time) was flawed and fixed 2 years later (Goldstein 1963). It was open for a long time whether a linear-time algorithm exists, before again Hopcroft and Tarjan (1974) presented their seminal algorithm. If we indeed have a planar graph, we are usually—in particular in most practical applications—interested in an embedding realizing a planar drawing. Hopcroft and Tarjan only sketched how to extract such an embedding from the data structures after the execution of the test, which Mehlhorn (1984) tried to clarify this with a more detailed description. Overall, it took 22 years for a crucial flaw—that becomes apparent when one tries to implement the algorithm—to be detected (and fixed) in this scheme (Mehlhorn and Mutzel 1996).

Runtime complexity versus runtime

The running time is the most commonly used criterion when evaluating algorithms and data structure operations. Here, asymptotically better algorithms are typically preferred over asymptotically inferior ones. However, usual performance guarantees are valid for all possible input instances, including pathological cases that do not appear in real-world applications. A classic example that shows how theoretically inferior algorithms may outperform their asymptotically stronger competitors in practice can be found in the field of mathematical programming: The most popular algorithm for solving linear programming problems—the simplex algorithm introduced by Dantzig in 1947—has been shown to exhibit exponential worst-case time complexity (Klee and Minty 1972), but provides very good performance for most input instances that occur in practice. In the 1970s, the polynomial-time solvability of linear programs was shown using the ellipsoid method (Khachiyan 1979), and promising polynomial alternatives have been developed since then. However, the simplex method and its variants dominated for decades due to their superior performance and are still the ones most widely used.

In practice, the involved constants—that are ignored in asymptotic analyses—play an important role. Consider again the history of planarity testing algorithms: the first linear time algorithm was presented in the 1970s (Hopcroft and Tarjan 1974); since this is the best possible asymptotical runtime, one could assume that the topic is therefore closed for further research. However, still nowadays new algorithms are developed for this problem. While the first algorithms were highly complex and required special sophisticated data structures and subalgorithms, the state-of-the-art algorithms (de Fraysseix and Ossona de Mendez 2003, Boyer and Myrvold 2004) are surprisingly simple and require nothing more than *depth first search* (DFS) traversals and ordered adjacency lists. Additionally, and also since they are comparably easy to implement, these new algorithms are orders of magnitude faster than the first approaches (Boyer et al. 2004).

An even more extreme example can be found in algorithms based upon the seminal graph minor theorem by Robertson and Seymour, discussed at length in the series *Graph Minors I–XX*. The theorem states that any minor-closed family of graphs is characterized by a finite *obstruction set*, i.e., a set of forbidden graph minors. In particular, the nonconstructive proof tells us that we can decide whether a given graph contains a fixed minor in cubic time. There are several graph problems, in particular many *fixed parameter tractable* (FPT) problems, that can be formulated adequately to use this machinery and obtain a polynomial algorithm. However, observe that the obstruction set, even when considered fixed, may in fact be very large and nontrivial to obtain. This leads to conceptual algorithms that, although formally polynomial, are of little to no use in practice.

An example is the FPT algorithm for the graph genus—i.e., to decide whether a given graph can be embedded on a surface of fixed genus g. The theorem tells us that this can be tested in polynomial time. However, even for the case $g = 1$—i.e. whether a graph can be drawn on the torus without crossings—the exact obstruction set is still unknown and has at least 16,629 elements, probably many more (Gagarin et al. 2005).

Generally, even though better asymptotic running time will typically pay off with growing input size, there are certain thresholds up to which a simpler algorithm may outperform a more complicated one with better asymptotic behavior. The selection of the best algorithm has to be made with respect to the characteristics of the input instances in the considered practical applications.

Traditionally, algorithmic analysis focuses on the required running times, while space consumption plays a second-order role. With the advent of mass data analysis,

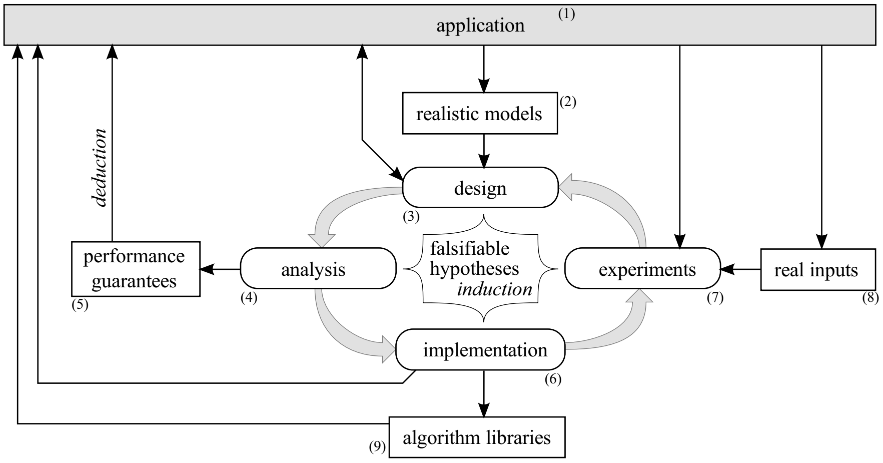


Fig. 6.1: The algorithm engineering cycle

e.g., whole genome sequencing, the latter concept gains importance. This is further aggravated by the fact that the increasing number of levels in the memory hierarchy leads to drastically different running times compared with what would be assumed based on the von Neumann model, see Sect. 6.3.2.

6.2 The Algorithm Engineering Cycle

There are multiple competing models that describe how software should be designed and analyzed. The most traditional one is the *waterfall model*, which describes the succession of design, implementation, verification, and maintenance as a sequential process. The algorithm engineering cycle—originally proposed by Sanders et al. (2005)—diverges from this view as it proposes multiple iterations over its substeps, cf. Fig. 6.1. It focuses not so much on software engineering aspects but rather on algorithmic development.

We usually start with some specific *application* (1) in mind and try to find a *realistic model* (2) for it such that the solutions we will obtain match the requirements as well as possible. The main cycle starts with an initial algorithmic *design* (3) on how to solve this model. Based on this design we *analyze* (4) the algorithm from the theoretical point of view, e.g., to obtain *performance guarantees* (5) such as asymptotic runtime, approximation ratios, etc. These latter two steps—to find and analyze an algorithm for a given model—are essentially the steps traditionally performed by algorithmic theoreticians in the pen-and-paper era.

We proceed with our algorithmic development by *implementing* (6) the proposed algorithm. We should never underestimate the usefulness of this often time-consuming process. It forms probably the most important step of algorithm engineering. First of all, we can only succeed in this step if the algorithm is reasonable in its implementation complexity. As noted before, often purely theoretical algorithms have been developed in the past, where it is unclear how to actually transform the theoretical ideas into actual code. Secondly, and in stark contrast to the assumption that theoretical proofs for algorithms are sufficient, it has often been the case that during this step certain flaws and omissions became obvious. As noted before, there have been papers which take theoretical algorithms and show that the runtime analysis is either wrong, because of some oversight in estimating the complexity of look-ups, or that the algorithmic description has to be extended in order to achieve the suggested running time.

Furthermore, algorithms may assume some static, probably preprocessed, data structure as its input, where certain queries can be performed asymptotically fast. However, in many real-world scenarios such input has to be stored in a more flexible format, e.g., in a linked list instead of an array. When thinking about graphs, this is even more common, as we will often store them as a dynamic graph with adjacency lists, or as compressed or packed adjacency matrices. In such data structures edge look-ups cannot be easily performed in constant time, as for a full adjacency matrix. However, the full unpacked matrix, which may be required by an algorithm to achieve optimal runtime performance, may simply be too large to be stored in practice. In such cases, alternative algorithms, which may be asymptotically slower under the assumption of constant-time edge look-ups but require fewer look-ups, may in fact be beneficial.

Such observations can also be made by corresponding *experiments* (7), which are the final major piece in our cycle. In particular, we are not interested in toy experiments, but in ones considering *real-world data* (8). Such input instances have to be at least similar to the problems the algorithm is originally designed for. The benefit of this step is manifold. It allows us to compare different algorithms on a testing ground that is relevant for the practice. We can better estimate the involved constants that have been hidden in the asymptotic runtime analysis, and thereby answer the question of which algorithm—even if asymptotically equivalent—is beneficial in practice. We may also find that certain algorithms may behave counter intuitively to the theoretical investigation: analytically slower algorithms may be faster than expected, either due to too crudely estimated runtime bounds or because the worst cases virtually never happen in practice. Such situations may not only occur in the context of running time, but are also very common for approximation algorithms, where seemingly weaker algorithms—probably even without any formal approximation guarantee—may find better solutions than more sophisticated algorithms with tight approximation bounds.

It remains to close the main algorithm engineering cycle. Our aim is to use the knowledge obtained during the steps of analysis, implementation, and experimentation to find improved algorithmic designs. E.g., when our experiments show a linear runtime curve for an algorithm with a theoretically quadratic runtime, this may give a hint for improving the theoretical analysis or that it is worthwhile to theoretically investigate the algorithm's average running time. Another approach is to identify bottlenecks of the algorithm during the experimentation, probably by profiling. We can then try to improve its real-world applicability by modifying our implementation or—even better—the algorithmic design.

Generally, during the whole cycle of design, analysis, implementation, and experimentation, we should allow ourself to also work on the basis of hypotheses, which can later be falsified in the subsequent steps.

A final by-product of the algorithm engineering cycle should be some kind of *algorithm library* (9), i.e., the code should be written in a reusable manner such that the results can be later used for further algorithmic development that builds upon the obtained results. A (freely available) library allows simpler verification of the academic findings and allows other research groups to compare their new results against preceding algorithms. Finally, such a library facilitates one of the main aspects of algorithm engineering, i.e., to bridge the gap between academic theory and practitioners in the nonacademic field. New algorithmic developments are much more likely to be used by the latter, if there exists a well-structured, documented reference implementation.

During the steps of this cycle, there are several issues that come up regularly, in particular because modern computers are only very roughly equivalent to the von Neumann machine. In the following section we will discuss some of these most common issues, which by now have often formed their own research fields.

6.3 Current Topics and Issues

When we move away from simple, cleanly defined problems and step into the realm of real-world problems, we are faced with certain inconsistencies and surprises compared with what theoretic analysis has predicted. This does not mean that the theory itself is flawed but rather that the considered underlying models do not exactly reflect the problem instances and the actual computer architecture. The inherent abstraction of the models may lead to inaccuracies that can only be detected using experimentation.

Generally, there are multiple issues that arise, rather independent of the specific problem or algorithm under investigation. In this section we try to group these into general categories and topics (cf. Fig. 6.2), which have by now defined worthwhile research fields of their own. Clearly, this categorization is not meant to be exhaustive but only representative. In the field of algorithm engineering we are always looking out for further interesting issues being revealed that influence algorithmic behavior in practice and are worth systematic investigation.

In the following we will concentrate on the algorithmic aspects that are involved when developing theoretical algorithms for real-world problems. Due to the algorithmic focus of this chapter, we will not discuss other modeling issues based on the fact that our underlying algorithmic problem might be too crude a simplification of the actual real-world problem. We will also not discuss the issues arising from noisy or partially faulty input data with which our algorithms may have to deal. However, bear in mind that these are in fact also critical steps in the overall algorithmic de-

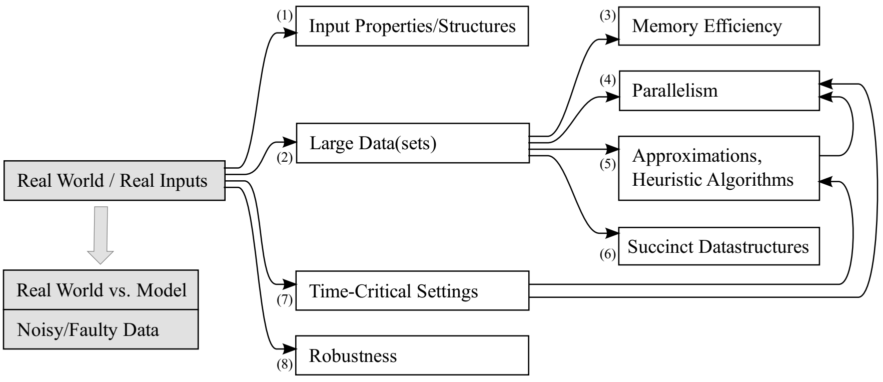


Fig. 6.2: Major topics and issues in current algorithm engineering

velopment. They may even influence the most fundamental algorithmic decisions, e.g., it will usually not be reasonable to concentrate on finding provably optimal solutions if the given input is too noisy.

6.3.1 Properties of and Structures in the Input

Often certain considerations regarding the expected input data are not regarded highly enough in purely theoretic analysis and evaluation of algorithms. Most analyses focus on worst-case scenarios, considering all inputs possible within the model's bounds. Average-case analyses are relatively rare, not only because they are mathematically much harder. It is hard to grasp what the "average case" should look like. Often—e.g., for the quicksort algorithm in the probably best known average-case analysis in algorithmics—it is assumed that all possible inputs are equally likely.

However, in most real-world scenarios the span of different inputs usually turns out to be only a fraction of all the possible inputs. In particular, the input data often has certain properties that are not problem but application specific. Failure to take these into account can lead to selecting inadequate algorithms for the problem at hand.

Consider problems on graphs. Many such problems become easier for graphs that are, e.g., planar or sparse, or have fixed tree width, fixed maximum degree, etc. Sometimes such properties even change the problem's complexity from *NP*-completeness to polynomially solvable. Alternatively, an algorithm may, e.g., find an optimal solution in such a restricted case, but only have a weak approximation ratio (if any) for the general case.

Clearly, when we know that our real-world data consists only of graphs of such a simplifying type, we ought to use algorithms specifically tuned for these kind of instances. However, algorithm engineering goes one step further: we ask the question of which algorithm will perform best, if we consider only graphs that are "close to," e.g., being planar. Due to the origin of the problem instances, such "close to" properties are often hard to grasp formally. Following the algorithm engineering cycle above we can either try to generalize special-purpose algorithms to be able to deal with such graphs, or to specialize general-purpose algorithms to deal with these graphs more efficiently. Often hybridization of these two approaches gives the best results in practice. After finding such efficient algorithms and demonstrating their performance experimentally, we can of course go back to theory and try to find some formal metric to describe this "close to" property and perhaps show some performance guarantees with respect to it.

In Sect. 6.4.1 we will showcase an example of such a development, regarding finding shortest paths in street maps. Even though the considered map graphs are not necessarily planar, we can still embed them in the plane with few crossings; although the edge weights do not satisfy the triangle inequality, the graphs are somehow close to being Euclidean graphs. This background knowledge allows for much more efficient algorithms in practice.

Interestingly, moving from the theoretical model to the practical one with respect to input instances does not always simplify or speed up the considered algorithms. In algorithmic geometry, there are many algorithms which assume that the given points are in *general position*, i.e., no three points may lie on a common line. It is often argued that, given there are some points which do not satisfy this property, one can slightly perturb their positions to achieve it. The other possibility is to consider such situations as special cases which can be taken care of by a careful implementation. The main algorithmic description will then ignore such situations and the implementer has to take care of them himself. Even though these additional cases may not influence the asymptotic runtime, they do affect the implementability and practical performance of the algorithm.

If we choose to perturb the points first—even disregarding any numerical problems that may arise from this step—we may lose certain important results which are in fact based on the collinearity of certain points. This is further amplified by the observation that real-world geometric data is often obtained by measuring points on some regular grid. Such input data can in fact exhibit the worst case for algorithms assuming general positions, but may on the other hand allow very simple alternative approaches. This shows that knowing the source and property of the input data can play a very important role in practice. However, no general rules can be applied or suggested, as the best approach is highly application specific.

6.3.2 Large Datasets

Huge data sets arise in many different research fields and are challenging not only because of the mere storage requirements, but more importantly because it is usually nontrivial to efficiently access, analyze, and process them.

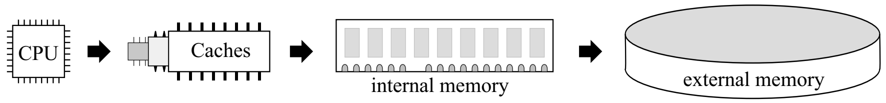


Fig. 6.3: Typical memory hierarchy in current PCs. Each component is orders of magnitude larger but also slower than the one on the previous layer

"One of the few resources increasing faster than the speed of computer hardware is the amount of data to be processed."
[IEEE InfoVis 2003 Call-For-Papers]

In bioinformatics, a single high-through put experiment can result in thousands of datapoints; since thousands of these experiments can be conducted per day, data sets with millions of entries can be quickly accumulated for further analysis. For example, the Stanford Microarray Database contains around 2.5 billion spot data from about 75,000 experiments (Demeter et al. 2007). Other fields of increasing importance are, among others, the analysis of the dynamics in telecommunication or computer networks, social statistics and criminalistics, and geographical information systems (GIS). The latter can, e.g., be used for computationally highly complex algorithms that analyze flooding or predict weather phenomena due to climate change.

The sheer size of such data sets can render traditional methods for data processing infeasible. New methodologies are therefore needed for developing faster algorithms when, e.g., even a simple look-up of the full data set is already inacceptably slow. This ranges from clever overall strategies to special-purpose heuristics, if the amount of data does not allow exact algorithms. When the data set is stable for a number of queries, such problems can often be tackled by preprocessing strategies whose computation costs are amortized over a number of optimization tasks. Also careless access to external memory storage such as hard disks may render the task at hand infeasible. For a realistic performance analysis in these cases, memory models that consider these costs have to be applied and algorithms have to be tuned to optimize their behavior respectively.

In the following, we will discuss some of these aspects in more detail.

6.3.3 Memory Efficiency

For a long time the von Neumann model was the dominant design model for computer architecture in the theoretical analysis of algorithms. An important aspect of this model—in contrast to current real-world computer hardware—is the assumption of a single structure storing both data and programs, allowing uniform constant memory access costs. In modern hardware architectures, the memory is organized in a hierarchy instead (cf. Fig. 6.3), where the access costs may differ between different levels by several orders of magnitude. Such memory hierarchies are mainly the result of a trade-off between speed and cost of available hardware.

Processors usually have a small number of internal registers that allow fast and parallel access but are expensive in terms of chip area. The next hierarchy layer is then made up of several (typically 2–3) levels of cache memory, currently multiple kilobytes to some megabytes in size. The main idea behind the use of cache memory is based on the assumption that memory access shows some temporal or spatial locality, i.e., objects are fetched from the same memory location within a short period of time or from locations that are close together. This allows copies of the data to be stored in fast memory for subsequent access, resulting in a *cache hit* if the data requested is stored in cache memory, and in a *cache miss* otherwise. The cache management, including which memory locations to store, where to store them, and which data to remove again, is controlled by the cache hardware and is usually outside the programmer's influence. When a cache miss occurs, the data has to be retrieved from the next memory level, the *internal (main) memory*, which is typically made up of relatively fast integrated circuits that are *volatile*, i.e., they lose the data information when powered off. The expensive access costs of lower memory levels are typically amortized by fetching a *block* of consecutive memory locations including the addressed one. The lowest level in the hierarchy, the *external memory*, provides large capacities, currently with gigabytes to terabytes of storage, and consists of nonvolatile memory that is much cheaper per byte than main memory. Current techniques include hard disks and more recently also so-called solid-state memory typically based on flash memory. The access to the external memory is called *input/output (I/O) operation* and may require blockwise or even sequential access. The type of access allowed and the access speed is often limited by the architecture of the memory hardware, e.g., hard disks require moving a mechanical read–write head, slowing down data retrieval significantly. In contrast the internal memory allows direct access to any memory locations in arbitrary order and is therefore also called *random access memory (RAM)*.

Performance guarantees built on the uniform access cost assumption therefore may not reflect the practical performance of the analyzed algorithms as the memory hierarchy may have a large impact. In order to approach a realistic performance rating, the analysis consequently needs to take the real memory organization into account. Clearly, it would not only be a very difficult task to analyze algorithms with respect to the exact architecture of existing systems, but also a futile attempt due to the constant change in this architecture. Hence, an abstraction of the existing memory hierarchies needs to be the basis of our memory model. This model should represent the main characteristics of real-world architectures and therefore allow the analysis to be close to the real behavior, but still be simple enough to be used with acceptable effort. The same argument can be used when designing algorithms: From an algorithm engineering perspective, a memory hierarchy model should already be applied in the algorithm design phase so that the real memory access costs can be minimized.

Several models have been proposed to allow performance analysis without having to consider particular hardware parameters. The conceptually most simple one is the *external memory model* (EMM) (Aggarwal and Vitter 1988), which extends the von Neumann model by adding a second, external, memory level. The M internal memory locations are used for the computation, whereas the external memory has to be accessed over expensive I/O operations that move a block of B contiguous words. Goals for the development of efficient external memory algorithms are to keep the number of internal memory operations comparable to the best internal memory algorithms while limiting the number of I/O operations. This can be done by implementing specific data replacement strategies to process as much data as possible in internal memory before writing them back and to maximize the amount of processable data in blocks that are read.

The field of external memory algorithms attracted growing attention in the late 1980s and the beginning of the 1990s, when increasingly large data sets had to be processed. A main contribution to the field was the work by Aggarwal and Vitter (1988) and Vitter and Shriver (1994a,b); an overview on external memory algorithms can be found in (Vitter 2001). Even though the external memory model is a strong abstraction of the real situation, it is a very successful model both in the amount of scientific research concerned with it as well as in the successful application of external memory algorithms in practice due to satisfactory performance estimation.

One could argue that the best way to consider, e.g., a cache memory level, would be to know exactly the hardware characteristics such as memory level size, block transfer size, and access cost, and then to consider these values during the algorithm's execution. Algorithms and data structures that are tuned in this way are called *cache-aware*. This approach not only needs permanent adaption and hinders an implementation on different hardware, but also complicates the analysis. In the late 1990s the theoretical concept of *cache-oblivious* algorithms was introduced (Frigo et al. 1999); it proposes better portable algorithms without having to tune them to specific memory parameters. Within this model, the parameters M and B are unknown to the algorithm, and in contrast to the EMM, block fetching is managed automatically by the hardware and the operating system.

It should be noted that, even though the term *cache* is used as a synonym for a specific level of the memory hierarchy, every level of the memory hierarchy can take the role of the cache for a slower level, and it is therefore sufficient to analyze cache-oblivious algorithms using a two-level hierarchy without compromising their performance on a deeper hierarchy. For multiple problems, including sorting, fast Fourier transformation, and priority queues, asymptotically optimal cache-oblivious algorithms have been found (Frigo et al. 1999).

Regarding effects on the performance induced by real hardware memory architecture, the actual picture is even more complicated than the discussion in this section. Not only is cache memory organized in multiple levels, but in multicore architectures that are mainstream today, caches on some of the levels might be shared between multiple cores whereas there are dedicated caches for each core on other levels. Also the internal cache design, the way the cache is operated, and hard-disk internal caches can have an impact on the resulting performance. Other influencing factors that we do not discuss here are effects related to physical organization and properties of the hardware, and to software such as the operating system and the compiler used.

A thorough discussion of memory hierarchies and the corresponding topics with respect to algorithm development is given by Meyer et al. (2003). The *standard template library for extra large data sets* (STXXL) provides an implementation of the C++ standard template library for external memory computations (Dementiev et al. 2008b).

6.3.4 Distributed Systems and Parallelism

Over the past few years we have seen remarkable advances regarding the availability of computing power. The main reason was not new advanced processor technology or increased clock speeds, but rather the availability of relatively cheap computers and the introduction of multicore processors that combine multiple processor cores on a single chip. Large numbers of computers can be clustered in farms at relatively low financial expense to tackle computational problems too complex to be solved with a single high-performance computer. Typical examples are render farms for *computer-generated imagery* (CGI), server farms for web searches, or computer clusters used for weather prediction.

In order to take advantage of the processing power these hardware environments provide, algorithm and data structure implementations need to be adapted. Programs have to be distributed over multiple processing units such that tasks, threads or instructions can be executed simultaneously to speed up computation. These processing units may be virtual processors allowing multithreading on a single processor, multiple cores on a chip, or multiple computers connected over a network, also in heterogeneous environments.

Parallelism can be exploited by *data decomposition* (i.e., distributing the computation among multiple threads processing different parts of data), *task decomposition* (i.e., operating on a set of tasks that can run in parallel), or a combination of both.

Typical issues regarding the performance of parallel processing are load balancing, i.e., distributing the workload to keep all processing units busy, data decomposition, task synchronization, and shared resource access, e.g., how to effectively use the shared bus bandwidth in multicore systems. When modeling parallel algorithms, also the communication costs between processors for synchronization etc. have to be taken into account. Considering distributed computing, we also have to cope with reliability issues, both of the hardware and the connection. This includes possible outages of processing units and consequently the tasks assigned to them, and connection management regarding latency times, information loss, etc. Together, all these factors comprise the parallelization overhead that limits the speed-up achieved by adding processing units, compared with the speed-up obtained for example by an increase of the clock rate.

A number of libraries exist that allow to exploit parallelism with low computational and programming overhead. The Open Multi Processing interface[1] supports multiplatform shared-memory parallel programming in C, C++, and Fortran and implements multithreading, where multiple sequential parts of a process, the threads, may share resources and are distributed among the processing units. Recently, a parallel implementation of the standard C++ library, the Multi-Core Standard Template Library (MCSTL), has been integrated into the GNU Compiler Collection (GCC) as the libstdc++ parallel mode. Internally, it uses OpenMP for multithreading and does not require specific source code changes to profit from parallelism. The open cross-platform parallel programming standard Open Computing Language[2] provides an API for the coordination of parallel computation and a C-based programming language with extensions to support parallelism to specify these computations.

Due to recent advances in the performance and programmability of modern graphics processing units (GPUs), the use of these for general-purpose computation is gaining increasing importance. Even when employed for graphics acceleration in standard PCs, their computational power is typically used only to a small extent most of the time. The availability of new interfaces, standardized instruction sets, and programming platforms such as OpenCL or the Compute Unified Device Architecture[3] to address graphics hardware allows to exploit this computational power and has led to a new field focused on the adaption of algorithm implementations in order to optimize their performance when processed on GPUs.

6.3.5 Approximations and Heuristic Algorithms

In traditional algorithmics, we tend to use the complexity classes of P and NP to decide what kind of algorithms to develop: if a problem is polynomially solvable, we try to find the (asymptotically or practically) fastest algorithm to solve the problem. If the problem is *NP*-hard, there cannot exist such algorithms (unless $P = NP$), and hence our efforts are divided into exact and inexact approaches. For the former, we allow that our runtime may become exponential in certain cases, but try to find algorithms which are “usually” much faster. For inexact approaches, we require a polynomial running time (probably also depending on parameters such as number of iterations etc.) but allow the solutions to be suboptimal. Approximation algorithms form a special subclass of such algorithms, as they guarantee that their solutions are mathematically close to the optimal (e.g., at most by a factor of 2 larger).

Recently, progress in computer hardware, modeling, and mathematical methods has allowed exact algorithms, based on integer linear programs, to run fast enough for practical applications (Polzin and Daneshmand 2001). In particular, there are even problem areas of combinatorial optimization where such approaches outper-

[1] Open Multi Processing API, `http://openmp.org/`

[2] Open Computing Language standard, `http://www.khronos.org/opencl/`

[3] Compute Unified Device Architecture, `http://www.nvidia.com/CUDA`

form state-of-the-art metaheuristics in terms of running time; see, e.g., (Chimani et al. 2009). Generally, such approaches typically are strong when the problem allows to include sophisticated problem-specific knowledge in the program formulation.

Identifying the best approach that fits the specific task at hand is clearly a critical decision and highly application dependent. Recent research advances show that traditional rules of thumb—e.g., "The problem is *NP*-hard, hence we have to resort to heuristics"—for this decision are often outdated.

Concerning the problem field of large datasets, we can even see a reverse situation than the one described above. Even though certain problems are polynomially solvable, the required runtime may be too large, regarding the amount of data to consider. Hence, we may need to develop heuristics or approximation algorithms with smaller runtime requirements, simply to be able to cope with the data in reasonable time at all.

A well-known example of such an approach is a method often used for *sequence alignment* in bioinformatics. Generally, we can compute the similarity of two sequences—textstrings representing, e.g., parts of genomes—in $\mathcal{O}(n \cdot m)$ time, whereby n and m are the lengths of the sequences. When investigating some unknown (usually short) sequence S of length n, the researcher wants to check it against a database of k (usually long) sequences, say of length m. Although we could find sequences similar to S in $\mathcal{O}(n \cdot m \cdot k)$ time, this becomes impractical when considering multigigabyte databases. Hence, usually only heuristic methods such as the well-known FASTA (Lipman and Pearson 1985, Pearson and Lipman 1988) or the *basic local alignment search tool* (BLAST) family (Korf et al. 2003) are used. Thereby the aim is to quickly discard sequences which have only a low chance of being similar to S (e.g., by requiring that at least a certain number of characters in the strings have to be perfect matches). The alignment itself can then also be approximated by only considering promising areas of the long sequences in the database.

6.3.6 Succinct Data Structures

In recent years, exact analysis of space requirements for data structures is gaining increasing attention due to the importance of processing large data sets. There is a strong need for space-efficient data structures in application areas that have to query huge data sets with very short response time, such as text indexing or storage of semistructured data.

This leads to the natural question how much space is needed not only to store the data but also to support the operations needed. In order to emphasize the focus on space efficiency, the term *succinct data structures* was coined for structures that represent the data with a space requirement close to the information-theoretic lower bound. Succinct data structures were introduced by Jacobson (1989) to encode bit vectors, trees, and planar graphs yet support queries efficiently. There has been a growing interest in this topic since then, leading to a wealth of publications that deal with both the practical and theoretical aspects of succinct data representations, see for example (Jansson et al. 2007, Geary et al. 2004, Golynski et al. 2007, Gupta 2007).

6.3.7 Time-Critical Settings

Above, we used the grand topic of large data sets to motivate the research topics regarding parallelism and also inexact approaches with approximate or heuristic solutions. Similar to this topic there is the large area of *time-critical* applications.

There are often cases where, although the amount of considered data is not too large to cause side-effects w.r.t. the memory hierarchy, it is impracticable to run traditional, probably optimal, algorithms on the full data set. E.g., a quadratic asymptotic runtime, even though not distressing on first sight, might turn out to be too much to solve a problem within tight time bounds. Such cases often arise when algorithms are used as subroutines within grander computing schemes, such that they are called a large number of times and even small performance benefits can add up to large speed-ups.

In other situations—in particular in real-time environments such as car systems, power-plant control systems, etc.—the situation may be even more critical: A real-time system sends a request to an algorithm and requires an answer within a certain timeframe. It can be a much larger problem if the algorithm misses this deadline, than if it would give a suboptimal—i.e., heuristic or approximative—answer.

We can view the research topics of parallelism (Sect. 6.3.4) and approximations (Sect. 6.3.5) also in the light of this problem field. While the overall concepts are similar, the developed algorithms have to cope with very different input data and external scenarios. In practice, this can lead to different algorithmic decisions, as certain aspects may become more relevant than others. In particular with parallelism, we have to keep a close look on the overhead introduced by many state-of-the-art techniques. E.g., on modern computer architectures, parallelization of sorting only becomes beneficial for a large number of elements (multiple millions); for fewer elements, traditional sorting is more efficient.

6.3.8 Robustness

As a final topic we want to touch on the area of algorithmic *robustness*, i.e., we aim for algorithms that behave stably in real-world scenarios. We differentiate between the two orthogonal problem fields regarding robustness: hardware failures and robustness with respect to input-data.

For the first category, we concentrate on the robust executability of algorithms. Even if our hardware exhibits failures—be it connection failures in distributed algorithms or random flips of bits in RAM modules—we would like to be able to detect and possibly recover from them. Especially the latter problem, though sounding fictitious at first, becomes a statistically important problem when considering large server farms, e.g., for web search (Henzinger 2004). It lead to the development of *resilient* algorithms where the aim is to find algorithms that are as efficient as the traditional ones, but can cope with up to k random bit flips (for some fixed k). Therefore we are not allowed to increase the memory requirement by the order of k, but only by at most $\omega(k)$; i.e., it is not an option to simply duplicate the memory sufficiently. See Finocchi and Italiano (2004) for a more detailed introduction to this topic, and first algorithms achieving this aim, e.g., for sorting data.

The second category of robustness deals with errors or slight modifications of the input data. Often, such input data will be a measured value with certain intrinsic inaccuracy. When we find a solution to our modeled problem, it is important to be able to apply it to the real-world scenario. Furthermore we would like that the found solution still resembles a good solution in this latter case, even if it turns out to be slightly different to the input used for the computation. Consider a shortest path problem where each edge length is only known within a certain margin of error. Assume there are two shortest paths with virtually the same lengths, one of which uses an edge with a high probability of being underestimated. Clearly, we would like our algorithm to find the other path that promises a shorter path with higher probability. For such scenarios, there are only a few results regarding purely combinatorial algorithms. However the problem of finding robust solutions with respect to small modifications to the input has been considered under the term *sensitivity analysis* in mathematical programming. Within mathematical programming, there are even dedicated research fields of *stochastic programming* and *robust optimization*, the first of which optimizes the average over the error distributions (or, in most cases, multiple *scenarios*), whereas the second optimizes the worst case. See for example Birge and Louveaux (2000) for an introduction to this topic.

6.4 Success Stories

Algorithm engineering has led to an improvement of algorithm and data structure performance in many different fields of application. We give two exemplary descriptions of the development of advanced approaches using algorithm engineering.

6.4.1 Shortest Path Computation

The history and still ongoing research on shortest path algorithms is probably the most prominent showcase of the successes of algorithm engineering. Interest in it has also been fueled by the 9th DIMACS implementation challenge[4] that asked for the currently best shortest path algorithms. We consider the traditional problem of finding a point-to-point (P2P) shortest path, i.e., given a graph with specified edge lengths, find the shortest edge sequence to connect two given nodes s and t of the graph. The problem is well known for decades, and often occurs as a subproblem within other complex problems. The presumably best-known application is in route-finding on street maps, railroad networks, etc.

In this realm, we can assume that all edge lengths are nonnegative, in which case we can use the famous algorithm Dijkstra presented already in (1959). Given a start node s, the algorithm in fact computes the shortest paths to all other nodes, but can be terminated early when the given target node t is scanned for the first time. The conceptual idea of the algorithm is as follows: during the course of the algorithm, we will label the nodes with upper bounds on their distances to s. Initially, all nodes except for s, which has distance 0, are labeled with $+\infty$. We hold all the nodes in a priority queue, using these labels as keys. We iteratively perform a *scanning* step, i.e., we extract the node v with smallest bound b. This bound then in fact gives the minimum distance from s to v. Now we update the bounds on the neighbors of v via *edge relaxation*: We decrease a label of a neighbor u to $b + d(v, u)$—where the latter term gives the length of the edge between v and u—if this value is less than the current label on u. Thereby we move u up in the priority queue correspondingly.

Hence the choice of the data structure realizing the priority queue becomes a major issue when bounding the algorithm's overall running time. Using a k-ary heap we can achieve $\mathcal{O}(m \log n)$, where n and m are the numbers of nodes and edges, respectively. By using a Fibonacci heap we can even guarantee $\mathcal{O}(m+n \log n)$ time.

Most interestingly, theoretic analysis suggests that the binary heap would constitute a bottleneck for sparse graphs. Since the considered graphs (street maps, railroad networks) often have this property, research has long concentrated on finding more efficient heap data structures. However, in practice, the really essential bottleneck is in fact not so much the complexity of the queue operations but the number of edge relaxations.

This insight and the investigation of the properties of the considered real-world networks led to algorithms far superior to Dijkstra's algorithm, although their theoretic runtime bounds are at best identical if not inferior. State-of-the-art P2P algorithms are *tens of thousands* of times faster than the standard Dijkstra approach. They usually require some seemingly expensive preprocessing before the first query, but these costs can be amortized over multiple fast queries. We can differentiate between a number of different conceptual acceleration ideas. Mostly, these concepts can be seen as orthogonal, such that we can combine acceleration techniques of different categories to obtain an even faster overall speed-up.

[4] DIMACS: Center for Discrete Mathematics and Theoretical Computer Science, `http://dimacs.rutgers.edu/`, Shortest Path Challenge: `http://www.dis.uniroma1.it/~challenge9/`

6.4.1.1 Bi- and Goal-Directed Algorithms

Dijkstra's algorithm scans all nodes that are closer to s than t. Although this property can come in handy in certain applications, it is the main reason why the algorithm is slow for P2P queries. The first improvements thereto were already proposed in the 1960s (bidirectional search) and the 1970s (goal-directed search). The former variant starts two Dijkstra algorithms from the start and the end node simultaneously. The search terminates when the two search frontiers touch, and we therefore have to scan fewer nodes, except for pathological cases.

The goal-directed (called A^*) search reduces the number of scanned nodes by modifying the keys in the priority queue: instead of only considering the distance between s and some node v for the key of v, it also takes estimates (in fact lower bounds) for the remaining distance between v and t into account. This allows us to extract nodes that are closer to t earlier in the algorithm and overall scan fewer nodes.

For graphs embedded in the Euclidean plane, we can simply obtain such lower bounds on the remaining distance geometrically. For general graphs, the concept of *landmarks* turned out to be useful (Goldberg and Harrelson 2005, Goldberg et al. 2005): for each node ℓ of some fixed subset of nodes, we compute the shortest distances to all other nodes in the preprocessing. These distances can then be used to obtain bounds for any other node pair.

6.4.1.2 Pruning

This acceleration technique tries to identify nodes that will not lie on the shortest paths and prune them early during the algorithm. This pruning may be performed either when scanning or when labeling a node. Multiple different concepts were developed, e.g., geometric containers (Wagner and Willhalm 2003), arc flags (Köhler et al. 2004), and reach-based pruning (Gutman 2004, Goldberg et al. 2005). Although they are all different, they use the common idea of preprocessing the graph such that, when considering a node or edge, we can—based on supplementing data structures—quickly decide whether to prune the (target) node and not consider it in the remaining algorithm.

6.4.1.3 Multilevel Approaches

The newest and in conjunction with the above techniques, most successful acceleration technique is based on extracting a series of graph layers from the input graph. The initial idea is to mimic how we manually would try to find an optimal route: when traveling between two cities, we will usually use city roads only in the beginning and at the end of our route, and use highways in between. The aim is to automatically extract such a hierarchy from the input graph, use it to accelerate the query times—by mainly using the smallest graph only considering the "fast" streets or subroutes—but still guarantee an overall optimal route. Multiple techniques arose to achieve this goal. While they are all beneficial to the more traditional approaches, the *highway hierarchies* as presented in Sanders and Schultes (2005) currently offer the best speed-ups.

6.4.1.4 State-of-the-Art and Outlook

Overall, for each such acceleration technique we can construct instances were they are not beneficial. However when applied to real-world instances such as the full road-networks of the USA or Europe, which contain multiple millions of nodes and edges, these techniques allow us to compute provable optimal shortest paths in a couple of microseconds (μs, a millionth part of a second). See, e.g., the repository of Schulz[5] for a comprehensive compilation of current algorithms and results.

Based on the successes for static route finding, current research now also focuses on adapting these algorithms for dynamic settings, taking into account situations where travel times on certain streets are dependent on the current time (e.g., rush hour versus 3 am) or on certain traffic conditions due to road blocks, accidents, etc.

6.4.2 Full-Text Indexes

Full-text indexes are data structures that store a (potentially long) text T and allow to search for a short substring, called the *pattern* p, within it. Finding all occurrences of p in T is an important task in many applications, e.g., for text search in large document collections or the Internet, data compression, and sequence matching in biological sequence databases.

The naive way to perform this task is to scan through the text, trying to match the pattern at each starting position, leading to a worst-case running time of $\mathcal{O}(nk)$, where $n = |T|$ and $k = |p|$. There are a number of direct search algorithms, such as those by Knuth–Morris–Pratt and Boyer–Moore, that try to decrease the necessary effort by skipping parts of the text, typically by preprocessing the pattern. Using this preprocessed information, these algorithms manage to achieve a search time of $\mathcal{O}(n + k)$, i.e., linear in the size of the text and the pattern. However, with today's huge data sets arising from high-throughput methods in biology or web indexing, even search time linear in the text size is inacceptably slow. Additionally, when the text is used for several searches of different patterns—e.g., when searching for peptide sequences in a protein database—it can pay off to build a data structure that uses information derived from the text instead, such that the construction time can be amortized over the number of searches. *Suffix trees* and *suffix arrays* are prominent examples of such data structures; after construction for a given input text they allow

[5] Frank Schulz, website collecting shortest path research, `http://i11www.iti.uni-karlsruhe.de/~fschulz/shortest-paths/`

to answer queries in time linear in the length of the pattern. Their concept is based on the fact that each substring of a text is also the prefix of a suffix of the text.

Depending on the task at hand and the available hard- and software environment, the time and space requirements for the construction of such a data structure may be critical. For many decades now, there has been an ongoing research into time- and space-efficient construction algorithms and implementations for suffix trees and arrays, especially motivated by the increasing needs in application areas such as bioinformatics.

In the 1970s, Weiner (1973) proposed a data structure to allow fast string searching—the *suffix tree*—that can be constructed in linear time and space in the length of the text string. The data is organized as a tree where paths from the root to the leaves are in a one-to-one correspondence to the suffixes of T. The suffix tree became a widely used string-matching data structure for decades. McCreight later improved the space consumption of the tree construction (1976), using about 25 % less than Weiner's original algorithm. Ukkonen (1995) improved on the existing algorithms by giving an online algorithm that allows the data structure to be easily updated.

With today's huge data collections, the memory consumption of suffix trees became a major drawback. Even though there have been many advances in reducing the memory requirements by developing space-efficient representations, see, e.g., Kurtz (1999), the use of 10–15 bytes on average per input character is still too high for many practical applications. The human genome, for example, contains about 3×10^9 base pairs, leading to a string representation of about 750 megabytes, whereas a suffix tree implementation would need around 45 gigabytes of internal memory. The performance of suffix trees is also dependent on the size of the alphabet Σ. When searching for DNA sequences, where $|\Sigma| = 4$, this is not a problematic restriction, but already for protein sequence searches with $|\Sigma| = 20$—and certainly when processing large alphabet texts such as, e.g., Chinese texts using an alphabet with several thousand characters—they are less efficient.

In the 1990s, Manber and Myers proposed a space-efficient data structure that is independent of the alphabet size—the *suffix array* (Manber and Myers 1993). It is constructed by sorting all suffixes of the text in lexicographic order and then storing the starting indices of the suffixes in this order in an array. This means that the look-up operation that returns a pointer to the i-th suffix in lexicographical order can be performed in constant time. The search for a pattern then can be done by using binary search in the array, leading to $\mathcal{O}(k \log n)$ runtime for a straight-forward implementation where a character-by-character comparison is done for each new search interval boundary. Using an additional array of longest common prefixes, the search can be sped up to $\mathcal{O}(k + \log n)$. Abouelhoda et al. showed how to use suffix arrays as a replacement for suffix trees in algorithms that rely on the tree structure and also discussed their application for genome analysis (Abouelhoda et al. 2002, 2004).

In recent years, there was a rally for the development of better, i.e., faster and more space-efficient, construction algorithms (Ko and Aluru 2005, Kim et al. 2005, Manzini and Ferragina 2004, Kärkkäinen et al. 2006). Today's best algorithms are fast in practice, some but not all of them have asymptotically optimal linear running time, they have low space consumption, and often can be easily implemented with only a small amount of code; see also Puglisi et al. (2007) for a classification and comparison of numerous construction algorithms.

6.4.2.1 Compressed and External Memory Indexes

Looking at the simplicity of the data structure and the great improvement in space consumption, the suffix array may appear to be the endpoint of the development. However, even the small space requirements of an integer array may be too large for practical purposes, e.g., when indexing biological databases, the data structure may not fit into main memory and in particular the construction algorithm may need far more space than is accessible. This motivated further research in two different directions: a further decrease in memory consumption and the development of external memory index construction algorithms. The first path led to so-called *compressed* data structures, whereas the second one led to algorithms with decreased I/O complexity.

Compressed data structures typically exhibit a trade-off between decreased space consumption for the representation and increased running time for the operations. When the compression leads to a size that allows to keep the whole data structure in internal memory, the increase in running time does not necessarily need to lead to slower performance in practice. Compressed indexes are especially well suited to reduce space complexity when small alphabet sizes, such as in genomic information, allow good compression rates. Compressed indexes that also allow to reproduce any substring without separate storage of the text, so-called *self-indexes*, can lead to additional space savings and are gaining increasing attention in theory and practice.

The first compressed suffix array, proposed by Grossi and Vitter (2005), reduced the space requirement of suffix arrays from $\mathcal{O}(n \log n)$ to $\mathcal{O}(n \log |\Sigma|)$. Sadakane extended its functionality to self-indexing (2003) and also introduced *compressed suffix trees* (2007), using only $\mathcal{O}(n \log |\Sigma|)$ bits but still providing the full functionality of suffix tree operations with a slowdown factor of polylog(n). Independently of the development of compressed suffix arrays, Mäkinen proposed the so-called *compact suffix array* (2002), which uses a conceptually different approach and is significantly larger than compressed suffix arrays, but is much faster to report pattern occurrences and text contexts.

In recent years, a number of improvements have been proposed that either decrease memory consumption or try to find a good practical setting regarding the space and time trade-off, e.g., the FM index (Ferragina and Manzini 2002, Ferragina et al. 2007), and compressed compact suffix arrays (Mäkinen and Navarro 2004, Navarro and Mäkinen 2007). Compressed indexes for the human genome can be constructed with a space requirement of less than 2 gigabytes of memory.

Arge et al. (1997) were the first to address the I/O complexity of the string sorting problem, showing that it is dependent on the relative lengths of the strings compared with the block size. Farach-Colton et al. (2000) presented the first I/O optimal construction algorithm, which is rather complex and hides large factors in the $\mathcal{O}$ notation. Crauser and Ferragina (2002) gave an experimental study on external memory construction algorithms, where they also described a discrepancy between worst-case complexity and practical performance of one of the algorithms and used their findings to develop a new algorithm that combines good practical and efficient worst-case performance. Dementiev et al. (2008a) engineered an external version of the algorithm by Kärkkäinen et al. (2006) that is theoretically optimal and outperforms all other algorithms in practice.

6.4.2.2 State-of-the-Art and Outlook

The research on text indexes led to many publicly available implementations, and the Pizza&Chili Corpus [6] provides a multitude of implementations of different indexes together with a collection of texts that represent a variety of applications. The available implementations support a common API and additional information. Software is provided to support easy experimentation and comparison of different index types and implementations.

Further improvements on the practical time and space consumption of the construction as well as the implementations of index structures can be expected, eventually for specific alphabets. Regarding the size of the texts and the availability of cheap medium-performance hardware, also the aspects of efficient use of external memory and of parallelization have to be further considered in the future. In order to speed up construction of text indexes for huge inputs, a number of parallel algorithms have been given (Kulla and Sanders 2007, Futamura et al. 2001). There are also parallel algorithms that convert suffix arrays into suffix trees (Iliopoulos and Rytter 2004).

6.5 Summary and Outlook

Algorithm engineering has come a long way from its first beginnings via simple experimental evaluations to become an independent research field. It is concerned with the process of designing, analyzing, implementing, and experimentally evaluating algorithms. In this chapter we have given an overview of some of the fundamental topics in algorithm engineering to present the main underlying principles. We did not touch too much on the topic of *designing* experiments, where the goal is to make them most expressive and significant. Though beyond the scope of this chapter, this is a critical part of the full algorithm engineering cycle; see, e.g., Johnson (2002) for a comprehensive discussion of this topic. More in-depth treatment of various other issues can be found in works by Mehlhorn and Sanders (2008), Dementiev (2006), and Meyer et al. (2003).

[6] Pizza&Chili Corpus, `http://pizzachili.dcc.uchile.cl/`

An increasing number of conferences and workshops are concerned with the different aspects of algorithm engineering, e.g., the International Symposium on Experimental Algorithms (SEA), the Workshop on Algorithm Engineering and Experiments (ALENEX), and the European Symposium on Algorithms (ESA). There are many libraries publicly available that provide efficient implementations of state-of-the-art algorithms and data structures in a variety of fields, amongst others the Computational Geometry Algorithms Library,[7] the Open Graph Drawing Framework,[8] and the STXXL for external memory computations. The Stony Brook Algorithm Repository[9] is a collection of algorithm implementations for over 70 of the most fundamental problems in combinatorial algorithmics.

Due to increasingly large and complex data sets and the ongoing trend towards sophisticated new hardware solutions, efficient algorithms and data structures will play an important role in the future. Experience in recent years, e.g., in the sequencing of the human genome, show that improvements along the lines of algorithm engineering—i.e., coupling theoretical and practical research—can lead to breakthroughs in application areas.

Furthermore, as demonstrated in the above sections, algorithm engineering is not a one-way street. By applying the according concepts, new and interesting questions and research directions for theoretical research arise.

References

Abouelhoda MI, Kurtz S, Ohlebusch E (2002) The enhanced suffix array and its applications to genome analysis. In: Proceedings of the Second International Workshop on Algorithms in Bioinformatics (WABI 02), Springer, pp 449–463

Abouelhoda MI, Kurtz S, Ohlebusch E (2004) Replacing suffix trees with enhanced suffix arrays. Journal of Discrete Algorithms 2(1):53–86

Aggarwal A, Vitter JS (1988) The input/output complexity of sorting and related problems. Communications of the ACM 31(9):1116–1127

Aho A, Johnson D, Karp RM, Kosaraju R, McGeoch C, Papadimitriou C (1997) Emerging opportunities for theoretical computer science. SIGACT News 28(3)

Arge L, Ferragina P, Grossi R, Vitter JS (1997) On sorting strings in external memory (extended abstract). In: Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC 97), ACM, pp 540–548

Auslander L, Parter SV (1961) On imbedding graphs in the plane. Journal of Mathematics and Mechanics 10(3):517–523

Birge JR, Louveaux F (2000) Introduction to Stochastic Programming. Springer

Boyer JM, Myrvold WJ (2004) On the cutting edge: Simplified $O(n)$ planarity by edge addition. Journal of Graph Algorithms and Applications 8(3):241–273

[7] Computational Geometry Algorithms Library, `http://www.cgal.org/`

[8] Open Graph Drawing Framework, `http://www.ogdf.net/`

[9] Stony Brook Algorithm Repository, `http://www.cs.sunysb.edu/~algorith/`

Boyer JM, Cortese PF, Patrignani M, Battista GD (2004) Stop minding your P's and Q's: Implementing a fast and simple DFS-based planarity testing and embedding algorithm. In: Proceedings of the 11th International Symposium on Graph Drawing (GD 03), Springer, LNCS, vol 2912, pp 25–36

Chimani M, Kandyba M, Ljubić I, Mutzel P (2009) Obtaining optimal k-cardinality trees fast. Journal of Experimental Algorithmics (Accepted). A preliminary version appeared in: Proceedings of the 9th Workshop on Algorithm Engineering and Algorithms (ALENEX 08), pages 27–36, SIAM, 2008.

Crauser A, Ferragina P (2002) A theoretical and experimental study on the construction of suffix arrays in external memory. Algorithmica 32(1):1–35

de Fraysseix H, Ossona de Mendez P (2003) On cotree-critical and DFS cotree-critical graphs. Journal of Graph Algorithms and Applications 7(4):411–427

Dementiev R (2006) Algorithm engineering for large data sets. PhD thesis, Universität des Saarlandes

Dementiev R, Kärkkäinen J, Mehnert J, Sanders P (2008a) Better external memory suffix array construction. Journal of Experimental Algorithmics 12:1–24

Dementiev R, Kettner L, Sanders P (2008b) Stxxl: standard template library for xxl data sets. Software: Practice & Experience 38(6):589–637

Demeter J, Beauheim C, Gollub J, Hernandez-Boussard T, Jin H, Maier D, Matese JC, Nitzberg M, Wymore F, Zachariah ZK, Brown PO, Sherlock G, Ball CA (2007) The Stanford microarray database: implementation of new analysis tools and open source release of software. Nucleic Acids Res 35(Database issue)

Dijkstra EW (1959) A note on two problems in connexion with graphs. Numerische Mathematik 1:269–271

Farach-Colton M, Ferragina P, Muthukrishnan S (2000) On the sorting-complexity of suffix tree construction. Journal of the ACM 47(6):987–1011

Ferragina P, Manzini G (2002) On compressing and indexing data. Tech. Rep. TR-02-01, University of Pisa

Ferragina P, Manzini G, Mäkinen V, Navarro G (2007) Compressed representations of sequences and full-text indexes. ACM Trans Algorithms 3(2):20

Finocchi I, Italiano GF (2004) Sorting and searching in the presence of memory faults (without redundancy). In: Proceedings of the 36th ACM Symposium on Theory of Computing (STOC 04), pp 101–110

Frigo M, Leiserson CE, Prokop H, Ramachandran S (1999) Cache-oblivious algorithms. In: Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS 99), IEEE Computer Society, p 285

Futamura N, Aluru S, Kurtz S (2001) Parallel suffix sorting. In: Proceedings of the 9th International Conference on Advanced Computing and Communications, Tata McGraw-Hill, pp 76–81

Gagarin A, Myrvold W, Chambers J (2005) Forbidden minors and subdivisions for toroidal graphs with no $K_{3,3}$'s. Electronic Notes in Discrete Mathematics 22:151–156

Geary RF, Raman R, Raman V (2004) Succinct ordinal trees with level-ancestor queries. In: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 04), SIAM, pp 1–10

Goldberg A, Harrelson C (2005) Computing the shortest path: A* search meets graph theory. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 05), SIAM

Goldberg AV, Kaplan H, Werneck R (2005) Reach for A*: Efficient point-to-point shortest path algorithms. Tech. Rep. MSR-TR-2005-132, Microsoft Research (MSR)

Goldstein AJ (1963) An efficient and constructive algorithm for testing whether a graph can be embedded in a plane. In: Proceedings of the Graph and Combinatorics Conference '63

Golynski A, Grossi R, Gupta A, Raman R, Rao SS (2007) On the size of succinct indices. In: Proceedings of the 15th Annual European Symposium on Algorithms (ESA 07), Springer, LNCS, vol 4698, pp 371–382

Grossi R, Vitter JS (2005) Compressed suffix arrays and suffix trees with applications to text indexing and string matching. SIAM Journal on Computing 35(2):378–407

Gupta A (2007) Succinct data structures. PhD thesis, Duke University, Durham, NC, USA

Gutman R (2004) Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In: Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX 04), SIAM, pp 100–111

Gutwenger C, Mutzel P (2001) A linear time implementation of SPQR-trees. In: Proceedings of the 8th International Symposium on Graph Drawing (GD 00), Springer, LNCS, vol 1984, pp 77–90

Henzinger M (2004) The past, present and future of web search engines (invited talk). In: 31st International Colloquium Automata, Languages and Programming, (ICALP 04), LNCS, vol 3142, p 3

Hopcroft JE, Tarjan RE (1973) Dividing a graph into triconnected components. SIAM Journal on Computing 2(3):135–158

Hopcroft JE, Tarjan RE (1974) Efficient planarity testing. Journal of the ACM 21(4):549–568

Iliopoulos CS, Rytter W (2004) On parallel transformations of suffix arrays into suffix trees. In: Proceedings of the 15th Australasian Workshop on Combinatorial Algorithms (AWOCA'04)

Jacobson G (1989) Space-efficient static trees and graphs. Symposium on Foundations of Computer Science 0:549–554

Jansson J, Sadakane K, Sung WK (2007) Ultra-succinct representation of ordered trees. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 07), SIAM, pp 575–584

Johnson DS (2002) A theoretician's guide to the experimental analysis of algorithms. In: Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges, AMS, pp 215–250

Kärkkäinen J, Sanders P, Burkhardt S (2006) Linear work suffix array construction. Journal of the ACM 53(6):918–936

Khachiyan LG (1979) A polynomial algorithm in linear programming. Dokl Akad Nauk SSSR 244:1093–1096, English translation in Soviet Math. Dokl. 20, 191-194, 1979

Kim DK, Sim JS, Park H, Park K (2005) Constructing suffix arrays in linear time. Journal of Discrete Algorithms 3(2-4):126–142

Klee V, Minty GJ (1972) How good is the simplex algorithm? In: Shisha O (ed) Inequalities III, Academic Press, pp 159–175

Knuth DE (1997) Art of Computer Programming, Volumes 1–3. Addison-Wesley Professional

Ko P, Aluru S (2005) Space efficient linear time construction of suffix arrays. Journal of Discrete Algorithms 3(2-4):143–156

Köhler E, Möhring RH, Schilling H (2004) Acceleration of shortest path computation. Article 42, Technische Universität Berlin, Fakultät II Mathematik und Naturwissenschaften

Korf I, Yandell M, Bedell J (2003) BLAST. O'Reilly

Kulla F, Sanders P (2007) Scalable parallel suffix array construction. Parallel Computing 33(9):605–612

Kurtz S (1999) Reducing the space requirement of suffix trees. Software–Practice and Experience 29:1149–1171

Lipman D, Pearson W (1985) Rapid and sensitive protein similarity searches. Science 227:1435–1441

Mäkinen V (2002) Compact suffix array: a space-efficient full-text index. Fundamenta Informaticae 56(1,2):191–210

Mäkinen V, Navarro G (2004) Compressed compact suffix arrays. In: Proceedings of the 15th Annual Symposium on Combinatorial Pattern Matching (CPM 04), Springer, LNCS, vol 3109, pp 420–433

Manber U, Myers G (1993) Suffix arrays: A new method for on-line string searches. SIAM Journal on Computing 22(5):935–948

Manzini G, Ferragina P (2004) Engineering a lightweight suffix array construction algorithm. Algorithmica 40(1):33–50

McCreight EM (1976) A space-economical suffix tree construction algorithm. Journal of the ACM 23(2):262–272

Mehlhorn K (1984) Graph algorithms and NP-completeness. Springer

Mehlhorn K, Mutzel P (1996) On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm. Algorithmica 16(2):233–242

Mehlhorn K, Sanders P (2008) Algorithms and Data Structures: The Basic Toolbox. Springer

Meyer U, Sanders P, Sibeyn JF (eds) (2003) Algorithms for Memory Hierarchies, LNCS, vol 2625. Springer

Navarro G, Mäkinen V (2007) Compressed full-text indexes. ACM Computing Surveys 39(1):2

von Neumann J (1993) First draft of a report on the EDVAC. IEEE Annals of the History of Computing 15(4):27–75

Pearson W, Lipman D (1988) Improved tools for biological sequence comparison. In: Proceedings of the National Academy of Sciences USA, vol 85, pp 2444–2448

Polzin T, Daneshmand SV (2001) Improved algorithms for the Steiner problem in networks. Discrete Applied Mathematics 112(1-3):263–300

Puglisi SJ, Smyth WF, Turpin AH (2007) A taxonomy of suffix array construction algorithms. ACM Computing Surveys 39(2):4

Sadakane K (2003) New text indexing functionalities of the compressed suffix arrays. Journal of Algorithms 48(2):294–313

Sadakane K (2007) Compressed suffix trees with full functionality. Theor Comp Sys 41(4):589–607

Sanders P, Schultes D (2005) Highway hierarchies hasten exact shortest path queries. In: Proceedings 17th European Symposium on Algorithms (ESA 05), Springer, LNCS, vol 3669, pp 568–579

Sanders P, Mehlhorn K, Möhring R, Monien B, Mutzel P, Wagner D (2005) Description of the DFG algorithm engineering priority programme. Tech. rep., DFG, `http://www.algorithm-engineering.de/beschreibung.pdf` (in German)

Ukkonen E (1995) On-line construction of suffix trees. Algorithmica 14(3):249–260

Vitter JS (2001) External memory algorithms and data structures: Dealing with massive data. ACM Computing Surveys 33(2):209–271

Vitter JS, Shriver EAM (1994a) Algorithms for parallel memory I: Two-level memories. Algorithmica 12(2/3):110–147

Vitter JS, Shriver EAM (1994b) Algorithms for parallel memory II: Hierarchical multilevel memories. Algorithmica 12(2/3):148–169

Wagner D, Willhalm T (2003) Geometric speed-up techniques for finding shortest paths in large sparse graphs. In: Proceedings of the 11th Annual European Symposium on Algorithms (ESA 03), Springer, LNCS, vol 2832, pp 776–787

Weiner P (1973) Linear pattern matching algorithms. In: Proceedings of the 14th Annual Symposium on Switching and Automata Theory (SWAT 73), IEEE Computer Society, pp 1–11

Part II
Characterizing Algorithm Performance

Chapter 7
Algorithm Survival Analysis

Matteo Gagliolo and Catherine Legrand

Abstract Algorithm selection is typically based on models of algorithm performance, learned during a separate *offline* training sequence, which can be prohibitively expensive. In recent work, we adopted an *online* approach, in which models of the runtime distributions of the available algorithms are iteratively updated and used to guide the allocation of computational resources, while solving a sequence of problem instances. The models are estimated using survival analysis techniques, which allow us to reduce computation time, censoring the runtimes of the slower algorithms. Here, we review the statistical aspects of our online selection method, discussing the bias induced in the runtime distributions (RTD) models by the competition of different algorithms on the same problem instances.

7.1 Introduction

This chapter is focused on modeling the performance of algorithms for solving *search* or *decision* problems (Hoos and Stützle 2004). In this broad category of problems, which includes satisfiability (Gent and Walsh 1999) and constraint satisfaction in general (Tsang 1993), there is no quality measure associated with a solution candidate, but only a binary criterion for distinguishing a solution: each problem instance may be characterized by zero, one or more solutions.

In the more general *optimization* problem, each solution is characterized by one or more scalar values, indicating its quality. While we do not explicitly consider

Matteo Gagliolo
IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium
University of Lugano, Faculty of Informatics, Lugano, Switzerland
e-mail: mgagliolo@iridia.ulb.ac.be

Catherine Legrand
Institut de Statistique, Université Catholique de Louvain, Louvain-la-Neuve, Belgium
e-mail: catherine.legrand@uclouvain.be

this category here, note that an optimization problem can be mapped to a decision problem if a target solution quality can be set in advance: in such cases, the corresponding decision problem consists of finding a solution with the desired quality level.

From a computer scientist's point of view, an algorithm is nothing but a piece of software, being executed on a particular machine. In this sense, its performance may be related to the use of various resources (memory, bandwidth, etc.), but the most widely used is definitely the CPU time spent, or *runtime*, and performance modeling is usually aimed at describing and predicting this value.

Consider then a randomized algorithm solving a given problem instance, and halting as soon as a feasible solution is found, or the instance is proved unsolvable. The runtime of the algorithm can be viewed as a random variable, which is fully described by its distribution, commonly referred to as the *runtime distribution* (RTD) in literature about algorithm performance modeling (see, e.g., Hoos and Stützle 2004).

Fortunately for us, we do not have to solve this performance modeling task from scratch. There is already a large corpus of research devoted to modeling the random occurrence of events in time: *survival analysis* (Klein and Moeschberger 2003, Collet 2003, Machin et al. 2006) is an important field of statistics, with applications in many areas such as medicine, biology, finance, technology, etc. Researchers in medicine will typically be interested in time to death (hence the name), but the analysis of *time-to-event* data in general occurs in other fields, and other terms may be used to describe the same thing. For example, engineers modeling the duration of a device speak of *failure analysis* or *reliability theory* (Nelson 1982). Actuaries setting premiums for insurance companies use the term *actuarial science*.

The rest of the chapter is organized as follows. Section 7.2 introduces the notions of survival analysis that can be of use when modeling runtime distributions. Section 7.3 describes the statistical aspects of our approach to algorithm selection (GAMBLETA) (Gagliolo and Schmidhuber 2006b, 2008a), focusing on the bias introduced in the models by the use of algorithm portfolios to sample the runtime distributions. Section 7.4 illustrates the presented concepts in a simple experimental setting. Section 7.5 references related work, and Sect. 7.6 concludes the chapter discussing ongoing research.

7.2 Modeling Runtime Distributions

In this section, we briefly review the basic concepts from survival analysis, and discuss their application to algorithm performance modeling.

7.2.1 Basic Quantities and Concepts

Let $T \in [0, \infty)$ be a random variable representing the time from a well-defined origin to the occurrence of a specific event: in our case, the *runtime* of an algorithm, defined as the interval between its starting time and its halting time, regardless of whether the algorithm halts because it found a solution or because it proved that there are none.

An important distinction has to be made between the RTD of a randomized algorithm on a given problem instance, which can be sampled by solving the instance repeatedly, each time initializing the algorithm with a different random seed, and the RTD of a randomized (or deterministic) algorithm on a set of instances, which can be sampled by running the algorithm repeatedly, each time solving a randomly chosen instance. In the first case, the random aspects are inherent to the algorithm. In the second, an additional random aspect is involved when picking the instance; another random process may be responsible for generating the instances in the set. In the following we will refer to these two distributions as the *RTD of the instance* and the *RTD of the set*, respectively; when no distinction is made, the concept exposed are valid in both cases.

A runtime distribution, as any other distribution, can be described by its *cumulative distribution function* (CDF),

$$F(t) = \Pr\{T \leq t\}, \;\; F : [0, \infty) \to [0, 1], \tag{7.1}$$

which is an increasing function of time, representing the probability that the algorithm halts within a time t. Another commonly used representation is the *probability density function* (pdf), defined as the derivative of the CDF:

$$f(t) = \frac{dF(t)}{dt}. \tag{7.2}$$

Integrating the pdf obviously gives us back the CDF:

$$F(t) = \int_0^t f(\tau) d\tau. \tag{7.3}$$

Time-to-event distributions are often described in terms of the *survival* function

$$S(t) = 1 - F(t), \tag{7.4}$$

which owes its name to the medical application of this branch of statistics, and represents, in our case, the probability that the algorithm will still be running at time t.

Another important concept in survival analysis is the *hazard* function $h(t)$, quantifying the instantaneous probability of the event of interest occurring at time t, given that it was not observed earlier:

$$h(t) = \lim_{\Delta t \to 0} \frac{\Pr\{t \leq T < t + \Delta t \mid T \geq t\}}{\Delta t} = \lim_{\Delta t \to 0} \frac{\Pr\{t \leq T < t + \Delta t\}}{\Delta t \Pr\{T \geq t\}} = \frac{f(t)}{S(t)}, \tag{7.5}$$

where $f(t)/S(t) = f(t \mid T \geq t)$ is the pdf conditioned on the fact that the algorithm is still running at time t.

The integral of (7.5) is termed the *cumulative* hazard, and has the following relationship with the survival function:

$$H(t) = \int_0^t h(\tau)d\tau = \int_0^t \frac{dF(\tau)}{S(\tau)} = -\ln S(t), \tag{7.6}$$

or conversely

$$S(t) = \exp(-H(t)). \tag{7.7}$$

The expected value of runtime, or *expected runtime*, can be evaluated as

$$E\{T\} = \int_0^\infty tf(t)dt = \int_0^1 tdF(t). \tag{7.8}$$

A *quantile* t_α of the RTD is defined as the time at which the probability $F(t)$ of the event reaches a value $\alpha \in [0, 1]$, and can be evaluated by solving the equation

$$t_\alpha = F^{-1}(\alpha). \tag{7.9}$$

A difference between the typical event of interest in survival analysis, death, and problem solution, is that the latter does not necessarily have to happen, as in general there is no guarantee that an algorithm will halt, whether by finding a solution or proving unsolvability of the instance at hand. This situation can characterize the behavior of an algorithm either on a single instance, if not all the runs halt, or on a set of instances, if the algorithm does not always halt for at least one of them. In both cases, the performance of the algorithm can be described by an *improper* RTD, where $F(\infty)$ is strictly smaller than unity, and represents the probability of halting. The integral of an improper pdf over $[0, \infty)$ does not sum to one:

$$\Pr\{T < \infty\} = F(\infty) = \int_0^\infty f(t)dt < 1 \tag{7.10}$$

the pdf for a successful run is

$$f(t \mid T < \infty) = \frac{f(t)}{F(\infty)}, \tag{7.11}$$

and the expected runtime is ∞, as:

$$E\{T\} = \Pr\{T < \infty\}E\{T \mid T < \infty\} + \Pr\{T = \infty\}E\{T \mid T = \infty\} \tag{7.12}$$

$$= F(\infty) \int_0^\infty tf(t \mid T < \infty)dt + [1 - F(\infty)]\infty = \infty, \tag{7.13}$$

while quantiles t_α are finite if $\alpha < F(\infty)$ infinite otherwise.

7.2.2 Censoring

The specific feature that makes survival analysis different from classical statistical analysis is *data censoring*, which refers to a situation in which the exact time of the event of interest is not known, but is known to lie in some (possibly open) interval. For example, in medical research analyzing survival of patients with a particular disease, some patient may still be alive at the time of performing the analysis of the data. The only information available about one of these subjects is that she has survived up to a certain time, which constitutes a lower bound on her survival time. Such incomplete observation of the event time is called *censoring*.[1]

Different types of censoring exist. In *type I* censoring, the event is observed only if it occurs before a prespecified censoring time, which can be different for each observation. *Type II* censoring occurs when we keep collecting data until a prespecified number of events has been observed. Quite often, these two types of censoring will result from experimenter's decisions aimed at reducing the duration of an experiment. Coming back to our medical example, one could decide to follow the patients for a predetermined period of time: all patients still alive at the end of this predetermined period will have a censored survival time. In this case, the duration of the experiment is fixed, while the number of observed deaths is a random variable. As an alternative, one could decide to end the experiment as soon as a predetermined number of deaths have been observed, the other patients' lifetimes being censored at the time the experiment ends. In this case, the number of observed events is fixed, while the duration of the experiment is a random variable. In other situations, the censoring mechanism is not controlled by the experimenter. This occurs quite frequently in medical applications, for example, when patients are "lost to follow-up" in the course of the study (i.e., they stop participating in it).

In any case, one has to be very careful with discarding incomplete data as this can result in an extremely biased model. Appropriate techniques have therefore been developed to take these censored observations into account, as we will see in the next subsection.

[1] More precisely, *right* censoring. One also speaks about *interval censoring* (we only know that the event occurred in between two time points without knowing when) and *left censoring* (we only know that the event occurred before a certain time). Right censoring is the most common, and is the only one we will consider here.

7.2.3 Estimation in Survival Analysis

In this section we consider the problem of estimating the survival distribution of an event, and modeling the impact of covariates on survival time, based on experimental data.

A sample of censored survival data is usually represented as a set of realizations $D = \{(t_1, c_1), \ldots, (t_n, c_n)\}$ of a pair of random variables (T, C), generated from another pair of *latent* random variables, the time to the actual event T_e and the time to censoring T_c. For each realization in the sample, only the minimum of these two variables $T = \min\{T_e, T_c\}$ is observed, along with the *event indicator* $C = I(T_e \leq T_c)$, which is 1 if the event of interest was observed and 0 if it was censored. Most estimation techniques require the time to event T_e and the time to censoring T_c to be statistically independent (*uninformative* censoring).

Methods for estimating a distribution can be broadly classified as parametric or nonparametric. In parametric methods, a parametric function is assumed to represent the time-to-event distribution. In nonparametric methods, no assumption is made on the distribution, and estimates are based solely on the observed data.

7.2.3.1 Parametric Methods

In this class of methods, a function $f(t; \boldsymbol{\theta})$ is assumed to represent the pdf of the runtime t. Several distributions are used to model survival times data, e.g., exponential, Weibull, log-normal, etc. The choice of a particular distribution is clearly a delicate issue, and should be based on empirical evidence: graphical diagnostic tests are available to evaluate a parametric model (Klein and Moeschberger 2003, Chap. 12).

The parameter $\boldsymbol{\theta}$ is estimated based on the data. Various approaches can be followed in this sense, usually based on the concept of the *likelihood* of the parameter in light of the data, that is, $L(\boldsymbol{\theta}; D) = \Pr\{D \mid \boldsymbol{\theta}\}$. In a *frequentist* context, *maximum-likelihood* methods estimate the parameter to be the one that maximizes the likelihood. In *Bayesian* approaches, the parameter value is assumed to be drawn from a *prior* distribution $\Pr\{\boldsymbol{\theta}\}$, and inference is based on the *posterior* probability $\Pr\{\boldsymbol{\theta} \mid D\} \propto \Pr\{D \mid \boldsymbol{\theta}\} \Pr\{\boldsymbol{\theta}\}$, either considering its mode as a point estimate of $\boldsymbol{\theta}$, or evaluating *credible intervals* based on the posterior (Bishop 1995, Mackay 2002). As both approaches require computing the likelihood, in the following we illustrate how this can be done in the case of censored survival data (Klein and Moeschberger 2003).

If $g(.)$ and $G(.)$ are, respectively, the pdf and the CDF of the censoring times, the contribution of a noncensored observation $(t_i, c_i = 1)$ to the likelihood is

$$L(\boldsymbol{\theta}; t_i, c_i = 1) = (1 - G(t_i)) f(t_i; \boldsymbol{\theta}), \tag{7.14}$$

while the contribution of a censored observation $(t_i, c_i = 0)$ is

$$L(\boldsymbol{\theta}; t_i, c_i = 0) = (1 - F(t_i; \boldsymbol{\theta}))g(t_i). \tag{7.15}$$

A sample of size n will be a combination of censored and noncensored observations, and assuming independence among the n realizations of (T, C), the likelihood will be given by

$$L(\boldsymbol{\theta}; D) = \prod_{i=1}^{n} [(1 - G(t_i))f(t_i; \boldsymbol{\theta})]^{c_i} [(1 - F(t_i; \boldsymbol{\theta}))g(t_i)]^{1-c_i}. \tag{7.16}$$

If we assume independent censoring (Liang et al. 1995, Fleming and Harrington 1991), the factors $(1 - G(t_i))$ and $g(t_i)$ are not informative for the inference on the parameters and can be removed from the likelihood, which can then be written

$$L(\boldsymbol{\theta}; D) \propto \prod_{i=1}^{n} f(t_i; \boldsymbol{\theta})^{c_i} (1 - F(t_i; \boldsymbol{\theta}))^{1-c_i}. \tag{7.17}$$

When the proposed parametric distribution for runtimes is appropriate, parametric is more efficient than nonparametric inference (Pintilie 2006), meaning that the resulting model will converge faster to the underlying distribution as the sample increases. Conversely, the use of a parametric function which does not fit the data well can produce an inefficient model.

7.2.3.2 Nonparametric Methods

If there is no censored data, a simple nonparametric estimate of the survival function can be obtained based on the empirical estimate of the CDF

$$\hat{F}(t) = \sum_{i:t_i \le t} \frac{1}{n}, \tag{7.18}$$

where $t_1 < t_2 < ... < t_r$ are the ordered survival times observed.

In the presence of censoring, the most commonly used nonparametric estimator of the survival function is the product-limit estimate proposed by Kaplan and Meyer (1958). This method is based on the idea of estimating the hazard at each time t_i in the sample as the portion of patients still alive, or “at risk” (in our case: the algorithms still running), experiencing an event at this time point:

$$\hat{h}(t_i) = \frac{\sum_{i:t_j = t_i, c_j = 1} 1}{\sum_{i:t_j \ge t_i} 1} = \frac{d_i}{n_i}, \tag{7.19}$$

where c_i is the event indicator, d_i is then the number of events observed at time t_i, and n_i is the number of observations still at risk at time t_i. An estimate of the survival function based on (7.19) is given by the *product limit* estimator, also known as the Kaplan-Meier (KM) estimator:

$$\hat{S}(t) = \prod_{i:t_i \le t} (1 - \hat{h}(t_i)) = \prod_{i:t_i \le t} \left(1 - \frac{d_i}{n_i}\right), \tag{7.20}$$

from which one can estimate the CDF as

$$\hat{F}(t) = 1 - \hat{S}(t). \tag{7.21}$$

The cumulative hazard can be obtained from (7.20) using the following relation:

$$\hat{H}(t) = -\log(\hat{S}(t)), \tag{7.22}$$

or based directly on (7.19), as proposed by Nelson (1972), Aalen (1978):

$$\hat{H}(t) = \sum_{i:t_i < t} \frac{d_i}{n_i}. \tag{7.23}$$

The two estimators are similar for large sample sizes.

In this and other nonparametric methods, $\hat{F}(t)$, $\hat{S}(t)$ and $\hat{H}(t)$ are stepwise functions that change their value only at uncensored observations $\{t_i \mid c_i = 1\}$ and are defined until the largest one; if the largest observation t_n is censored, the resulting estimate $\hat{F}(t_n)$ will be < 1. The hazard estimate $\hat{h}(t)$ (7.19) is discrete, defined only at the event times $\{t_i\}$ in the sample D. In order to obtain meaningful predictions also for $t \notin \{t_i\}$, hazard estimates can be *smoothed* (Wang 2005).

A review of nonparametric estimation for censored survival data in a Bayesian framework can be found in (Ibrahim et al. 2001).

7.2.3.3 Regression Models

If additional information is available about each observation, in the form of some *covariates* or *features* $\mathbf{x} \in \mathbb{R}^d$, it may be of interest to investigate its relationship with the time to event. This can be done estimating a conditional model, or *regression* model, of the survival distribution $S(t \mid \mathbf{x})$ or, as is quite often done, of the baseline hazard $h(t \mid \mathbf{x})$. In our case, consider again the RTD of a set of instances, and the RTDs of each instance in the set, which will in general be different from the RTD of the set. When performing algorithm selection, with the aim of reducing runtime, it is clearly not advisable to model the RTD of each instance, as this would require obtaining multiple runtimes for each instance. As we expect similar instances to have similar RTDs, we could instead identify features of the problem instance that can be related to the runtime of the algorithm, in order to recover estimates that are closer to the actual RTD of each instance. In another situation, one might want to study the impact of the various parameters of an algorithm on its runtime distribution: in this case a sample could be formed using different values of the parameters, and the parameters could be used as covariates.

A parametric regression model can be based on a pdf $f(t \mid \boldsymbol{\theta})$ in which the parameter is itself a parametric function of the covariate $\boldsymbol{\theta}(\mathbf{x}; \boldsymbol{\beta})$ (see, e.g., Bishop

1995, par. 6.4): for example, an exponential distribution $f(t;\theta) = \theta \exp(-\theta t)$ can be cascaded with a linear model $\theta(\mathbf{x};\boldsymbol{\beta}) = \boldsymbol{\beta} \cdot \mathbf{x}$, obtaining a parametric conditional model:

$$f(t \mid x;\boldsymbol{\beta}) = \theta(\mathbf{x};\boldsymbol{\beta}) \exp(-\theta(\mathbf{x};\boldsymbol{\beta})t) = (\boldsymbol{\beta} \cdot \mathbf{x}) \exp(-\boldsymbol{\beta} \cdot \mathbf{x}). \tag{7.24}$$

Various other parametric models have been proposed to study the dependency between covariates and the survival time. Several of them model this dependency via the hazard or the cumulative hazard function, and the most popular one is probably the *proportional hazards* model, or Cox model (Cox 1972):

$$h(t \mid \mathbf{x}) = h_0(t) \exp(\boldsymbol{\beta} \cdot \mathbf{x}). \tag{7.25}$$

In this model it is assumed that the covariates act multiplicatively on a baseline hazard $h_0(t)$: if this is left unspecified, (7.25) is termed a *semiparametric* model. With such a model, the ratio of hazards of two observations is independent of time, and is determined by the corresponding covariates. Among other (semi)parametric models, one can cite the additive risk model (Klein and Moeschberger 2003, Chap. 10), in which the covariates act additively on the hazard function, and the accelerated failure time model (Cox and Oakes 1984), in which the covariates are related to the logarithm of survival times.

Only limited work has been done on fully nonparametric regression models (see the reviews by Keilegom et al. 2001, Spierdijk 2005). Beran (1981) and later works (e.g., Akritas 1994, Wichert and Wilke 2005) are based on the idea of estimating a conditional hazard function, as in (7.19), but with 1 replaced by a *kernel* function $K(\mathbf{x},\mathbf{y}) : \mathbb{R}^d \times \mathbb{R}^d \to [0,\infty)$ representing a similarity measure in covariate space: a prediction of the hazard at time t for an individual with covariate $\mathbf{x}$ is given by

$$\hat{h}(t \mid \mathbf{x}) = \frac{\sum_{i:t_i=t,c_i=1} K(\mathbf{x},\mathbf{x}_i)}{\sum_{j:t_j \geq t} K(\mathbf{x},\mathbf{x}_j)}. \tag{7.26}$$

In all these conditional models, the covariates are constant over time. In some situations, it might be interesting to model the effect of covariates which vary over time. In our case, *dynamic* information about the algorithm could be available, for example, the state of some internal variables. One possibility is to treat it as a *time-varying* covariate (Li and Doss 1995, Nielsen and Linton 1995); another possibility is to consider *longitudinal data* analysis techniques (Fitzmaurice et al. 2008).

7.2.4 Competing Risks

In some situations, occurrence of another type of event may "compete" with the event of interest, and prevent the observation of the latter for some individuals. For example, in a medical setting, one might want to study the distribution of time to death T_A, due to a particular disease or condition A, and gather a sample of individuals affected by it: at the end of the study, some patients will have died of condition A, and their lifetime recorded as uncensored, while others will be still alive, and their time to death censored. However, some of the patients might have died from a different condition, B. If we want to estimate the distribution of time to death of A, we may view death of B as a censoring event, as it prevented us from knowing what T_A *would* have been if the patient did not die of B at a time $T_B < T_A$. Death from B is termed a *competing risk* (Pintilie 2006, Putter et al. 2006).

As death of B is an event that we are not interested in modeling, one possibility is to consider it as an additional censoring mechanism. This may pose a problem when the distribution of the event of interest and that of the competing risk are not independent, for example, if both conditions A and B become more likely with old age. This would obviously result in a censoring mechanism that is not independent from the distribution of time to events, thus not satisfying assumption on which most estimators rely.

When modeling runtime distributions, this situation can generally be avoided, as one can always "kill" the same "patient" (i.e., the same problem instance) multiple times, using different algorithms independently, as different "causes of death". This is not the case with our online algorithm selection method (Section 7.3), where we strive to avoid unnecessary computations, solving each instance only once, and using the fastest algorithm as a censoring mechanism for sampling runtimes of the slower ones; so the topic discussed in this subsection is more relevant to our work, rather than to RTD modeling in general.

Consider K competing risks, represented by K random variables $T_1, \dots, T_K$, each with its own distribution, sampled with some censoring mechanism with threshold T_c. For each individual, we only observe a pair of random variables (T, C), where $T = \min\{T_1, \dots, T_K, T_c\}$ is the event time, and the event indicator C is 0 if the individual is still alive at T_c and k if the individual died of the k-th risk, so it is a discrete random variable with value in $\{0, 1, \dots, K\}$.

Consider the *joint survival function* of the times T_k

$$Z(t_1, \dots, t_K) = \Pr\{T_1 > t_1, \dots, T_K > t_k\}. \tag{7.27}$$

The probability of the event $\{T_k > t_k\}$ is the *marginal* survival function

$$\Pr\{T_k > t_k\} = Z_k(t_k) = Z(0, \dots, t_k, \dots, 0). \tag{7.28}$$

If the event times $\{T_k\}$ are independent, the joint probability (7.27) equals the product of the probabilities of the single events (7.28):

$$Z(t_1, \dots, t_K) = \prod_{j=1}^{K} Z_j(t_j). \tag{7.29}$$

Unfortunately, if the data is gathered as described above, with at most one uncensored event time per individual, the independence assumption cannot be tested.

Neither the joint survival function (7.27) nor the marginals (7.28) can be identified, and Kaplan-Meier estimates of the marginals for each risk will be biased (Tsiatis 1975, Putter et al. 2006).

One can always estimate the overall survival probability for an individual,

$$S(t) = \Pr\{T_1 > t, \ldots, T_K > t\} = Z(t_1 = t, t_2 = t, \ldots, t_K = t), \tag{7.30}$$

with a product limit estimate (7.20) from the data, considering all recorded events as if they were the same event. Other quantities that can be estimated (Pintilie 2006, Putter et al. 2006) are the *cause-specific hazard*, which is defined as the hazard of failing for a specific cause k:

$$\lambda_k(t) = \lim_{\Delta t \to 0} \frac{\Pr\{t \le T < t + \Delta t, C = k \mid T \ge t\}}{\Delta t}, \tag{7.31}$$

from which one can obtain the cumulative cause-specific hazard,

$$\Lambda_k(t) = \int_0^t \lambda_k(\tau) d\tau, \tag{7.32}$$

and the *subsurvival* function

$$R_k(t) = \exp(-\Lambda_k(t)), \tag{7.33}$$

with

$$S(t) = \prod_k R_k(t) = \exp(-\sum_k \Lambda_k(t)). \tag{7.34}$$

The *cumulative incidence function* or *subdistribution function* for the cause k is defined as the probability of failing from cause k before time t:

$$I_k(t) = \Pr\{T \le t, C = k\} = \int_0^t \lambda_k(\tau) S(\tau) d\tau. \tag{7.35}$$

This is an improper distribution, as $I(\infty) = \Pr\{C = k\} = \Pr\{T_j > T_k \forall j \neq k\}$, which can be smaller than 1.

7.3 Model-Based Algorithm Selection

Attempts to automate *algorithm selection* are not new (Rice 1976) and are motivated by the fact that alternative algorithms for solving the same problem often display different performance on different problem instances, such that there is no single "best" algorithm. Selection is typically based on predictive models of algorithm performance, which are either assumed to be available, or are learned during an initial training phase, in which a number of problem instances are solved with each of the available algorithms, in order to sample their performances.

As we have seen, the natural performance model for a decision problem solver is its runtime distribution. The advantage of survival analysis techniques is that they allow to reduce the time spent of sampling the RTD, as excessively long runs of the algorithm can be censored, and still contribute to the model. It is of note that there is an obvious *trade-off* between the computational cost of the sampling experiment and the portion of uncensored events observed, which in turn will have an impact on the precision of the obtained model: a higher portion of uncensored events will imply a larger time cost, but it will also result in a more precise estimate of model parameters. In the context of algorithm selection techniques, this trade-off should rather be measured between training time and the gain in performance resulting from the use of the learned model: in this sense, the required model precision can be much lower than expected. An example of such a situation is given by (Gagliolo and Schmidhuber 2006a) where this trade-off is analyzed in the context of restart strategies,[2] reporting the training times, and the resulting performance, of a model-based restart strategy, where the RTD model is learned with different censoring thresholds. Inspired by this idea, Gagliolo and Schmidhuber (2007) proposed an *online* method for learning a restart strategy (GAMBLER), in which a model of the RTD of an algorithm is iteratively updated *and* used to evaluate an optimal restart strategy for the algorithm. Gagliolo and Schmidhuber (2006b, 2008a), adopted a similar online approach (GAMBLETA) to allocate computational resources to a set of alternative algorithms. In both cases, the trade-off between exploration and exploitation is represented as a *multi-armed bandit problem* (Auer et al. 2003). This game consists of a sequence of trials against a K-armed slot machine: at each trial, the gambler chooses one of the available arms, whose losses are randomly generated from different unknown distributions, and incurs of the corresponding loss. The aim of the game is to minimize the *regret*, defined as the difference between the cumulative loss of the gambler and that of the best arm.

In GAMBLETA (Gagliolo and Schmidhuber 2006b, 2008a), the different "arms" of the bandit are represented by different *time allocators* (TA), that is, different techniques for allocating computation time to the available algorithms. To understand the rationale for this approach, consider a situation in which we have a way to allocate time based on a model of the RTDs of the algorithms, i.e., a *model-based* TA. In an online setting, in which the models are gradually improved as more instances are solved and a larger runtime sample is available, the performance of this TA will obviously be very poor at the beginning of the problem sequence. This model-based TA can be combined with a more exploratory one, such as a parallel portfolio of all available algorithms, and, at an upper level, the bandit problem solver (BPS) can be used to pick which TA to use for each subsequent problem instance, based on the previous performances of each TA. Intuitively, the BPS will initially penalize the model-based allocator, but only until the model is good enough to outperform the exploratory allocator. Alternative allocation techniques can be easily added as additional "arms" of the bandit.

[2] A restart strategy consists of executing a sequence of runs of a randomized algorithm in order to solve a given problem instance, stopping each run j after a time t_j if no solution is found and restarting the algorithm with a different random seed.

To introduce some notation, consider a portfolio $\mathcal{A} = \{a_1, a_2, ..., a_K\}$ of K algorithms, which can be different algorithms, different parametrization of the same algorithm, or even copies of the same randomized algorithm differing only in the random seed, or any combination thereof. The runtime of each a_k on the current problem instance is a random variable, T_k. The algorithms solve the same problem instance in parallel, and share the computational resources of a single machine according to a *share* $\mathrm{s} = \{s_1, .., s_K\}, s_k \geq 0, \sum_{i=1}^{K} s_k = 1$; i.e., for any amount t of machine time, a portion $t_k = s_k t$ is allocated to a_k. Here and in the following we assume an "ideal" machine, with no task-switching overhead. Algorithm selection can be represented in this framework by setting a single s_k value to 1, while a uniform algorithm portfolio would have $\mathrm{s} = (1/K, ..., 1/K)$.

A time allocator (TA) can be defined as a device setting the share s for the algorithms in the portfolio, possibly depending on the problem instance being solved. A *model-based* TA can be implemented mapping the estimated RTDs of the algorithms on the current instance to a share value s. Gagliolo and Schmidhuber (2006b) proposed three different analytic approaches, in which time allocation is posed as a function optimization problem: the RTD of a portfolio is expressed as a function of the share s, and the value of s is set optimizing some function of this RTD, for example, the expected runtime. The next sections will illustrate these two steps.

7.3.1 RTD of an Algorithm Portfolio

An a_k that can solve the problem in time t_k if run alone will spend a time $t = t_k/s_k$ if run with a share s_k. If the runtime distribution $F_k(t_k)$ of a_k on the current problem is available, one can obtain the distribution $F_{k,s_k}(t)$ of the event "a_k solved the problem after a time t, using a share s_k", by simply substituting $t_k = s_k t$ in F_k:

$$F_{k,s_k}(t) = F_k(s_k t). \tag{7.36}$$

If the execution of all the algorithms is stopped as soon as one of them solves the problem, the resulting duration of the solution process is a random variable, representing the runtime of the parallel portfolio, whose value is determined by the share s, and by the random runtimes T_k of the algorithms in the set:

$$T_{\mathcal{A}} = \min\left\{\frac{T_1}{s_1}, \frac{T_2}{s_2}, \ldots, \frac{T_K}{s_K}\right\}. \tag{7.37}$$

Its distribution $F_{\mathcal{A},\mathrm{s}}(t)$ can be evaluated based on the share s, and the $\{F_k\}$. The evaluation is more intuitive if we reason in terms of the survival distribution: at a given time t, the probability $S_{\mathcal{A},\mathrm{s}}(t)$ of *not* having obtained a solution is equal to the joint probability that *no single algorithm* a_k has obtained a solution within its time share $s_k t$. Assuming that the solution events are independent for each a_i, this joint probability can be evaluated as the product of the individual survival functions $S_k(s_k t)$

$$S_{\mathcal{A},\mathbf{s}}(t) = \prod_{k=1}^{K} S_k(s_k t). \tag{7.38}$$

Given (7.6), equation (7.38) has an elegant representation in terms of the cumulative hazard function. Apart from the terms s_k, (7.39) is the method used by engineers to evaluate the failure distribution of a *series* system, which stops working as soon as one of the components fail, based on the failure distribution for each single component (Nelson 1982):

$$H_{\mathcal{A},\mathbf{s}}(t) = -\ln(S_{\mathcal{A},\mathbf{s}}(t)) = \sum_{k=1}^{K} -\ln(S_k(s_k t)) = \sum_{k=1}^{K} H_k(s_k t). \tag{7.39}$$

The above formulas rely on the assumption of independence of the runtime values among the different algorithms, which allows the joint probability (7.38) to be expressed as a product. This assumption is met only if the S_k represent the runtime distributions of the a_k on the particular problem *instance* being solved. If instead the only S_k available capture the behavior of the algorithms on a *set* of instances, which includes the current one, independence cannot be assumed: in this case, the methods presented should be viewed as heuristics. In a less pessimistic scenario, one could have access to models of the S_k conditioned on features, or *covariates*, $\mathbf{x}$ of the current problem. In such a case the *conditional* independence of the runtime values would be sufficient, and the resulting joint survival probability could still be evaluated as a product

$$S_{\mathcal{A},\mathbf{s}}(t \mid \mathbf{x}) = \prod_{i=1}^{K} S_k(s_k t \mid \mathbf{x}). \tag{7.40}$$

7.3.2 Model-Based Time Allocators

Once the RTD of the portfolio is expressed as a function of the share $\mathbf{s}$, as in (7.38, 7.40), the problem of allocating machine time can be formulated as an optimization problem. In (Gagliolo and Schmidhuber 2006b), the following alternatives are proposed:

Expected time The expected runtime value $E_{\mathcal{A},\mathbf{s}}(t) = \int_0^\infty t f_{\mathcal{A},\mathbf{s}}(t)dt$ can be obtained, and minimized with respect to $\mathbf{s}$:

$$\mathbf{s} = \arg\min_{\mathbf{s}} E_{\mathcal{A},\mathbf{s}}(t). \tag{7.41}$$

Contract If an upper bound, or *contract*, t_u on runtime is imposed, one can instead use (7.38) to pick the $\mathbf{s}$ that maximizes the probability of solution within the contract $F_{\mathcal{A},\mathbf{s}}(t_u) = \Pr\{T_{\mathcal{A},\mathbf{s}} \leq t_u\}$ (or, equivalently, minimizes $S_{\mathcal{A},\mathbf{s}}(t_u)$):

$$\mathbf{s} = \arg\min_{\mathbf{s}} S_{\mathcal{A},\mathbf{s}}(t_u). \tag{7.42}$$

Quantile In other applications, one might want to solve each problem with probability at least α, and minimize the time spent. In this case, a quantile $t_{\mathcal{A},\mathbf{s}}(\alpha) = F^{-1}_{\mathcal{A},\mathbf{s}}(\alpha)$ should be minimized:

$$\mathbf{s} = \arg\min_{\mathbf{s}} F^{-1}_{\mathcal{A},\mathbf{s}}(\alpha). \tag{7.43}$$

These three methods can be easily adapted to allocate time on multiple CPUs (Gagliolo and Schmidhuber 2008b). If the F_k are parametric, a gradient of the above quantities can be computed analytically, depending on the particular parametric form; otherwise, the optimization can be performed numerically.

Note that the shares $\mathbf{s}$ resulting from these three optimization processes can differ: in the last two cases, they can also depend on the values chosen for t_u and α respectively. In no case is there a guarantee of unimodality, and it may be advisable to repeat the optimization process multiple times, with different random initial values for $\mathbf{s}$, in case of extreme multimodality.

7.3.3 Algorithms as Competing Risks

As said, GambleTA is a fully online algorithm selection method, in which there is no distinction between learning and using the RTD models. In order to save computation time, for each instance, we only wait until the fastest algorithm solves it: at this point we stop the execution of the remaining algorithms. In medical terms, we are viewing each instance b_j as a patient, with covariates $\mathbf{x}_j$, and the K algorithms as competing risks, one of which should eventually "kill" the patient, and censoring the runtime values for the other algorithms.

From a statistical point of view, a clear disadvantage of using an algorithm portfolio to gather runtime data is that, as we saw in Sect. 7.2.4, the resulting models will be biased. In the next section we will see an illustrative example of this phenomenon.

7.4 Experiments

In this section we present a simple sampling experiment, with a small set of satisfiability problem solvers, in order to illustrate the concepts introduced in this chapter.

Satisfiability (SAT) problems (Gent and Walsh 1999) are standard decision problems, with many important practical applications. A conjunctive normal form CNF(k,n,m) problem consists of finding an instantiation of a set of n Boolean variables that simultaneously satisfies a set of m clauses, each being the logical OR of k literals, chosen from the set of variables and their negations. A problem instance is termed *satisfiable* (SAT) if there exists at least one such instantiation, otherwise it is *unsatisfiable* (UNSAT). An algorithm solving an instance will halt as soon as a single solution is found or unsatisfiability is proved. With $k = 3$ the problem is NP-complete. Satisfiability of an instance depends in probability on the ratio of clauses to variables: a *phase transition* can be observed at $m/n \approx 4.3$ (Mitchell et al. 1992), at which an instance is satisfiable with probability 0.5. This probability quickly goes to 0 for m/n above the threshold, and to 1 below.

SAT solvers can be broadly classified in two categories: *complete* solvers, which execute a backtracking search on the tree of possible variable instantiations and are guaranteed to determine the satisfiability of a problem in a finite, but possibly unfeasibly high, amount of time; and *local search* (LS) solvers, that cannot prove unsatisfiability, but are usually faster than complete solvers on satisfiable instances. In other words, a local search solver can only be applied to satisfiable instances: at the threshold, there is a 0.5 probability that the instance will be UNSAT, in which case the solver would run forever. The RTD of a set of instances for a complete solver will have $F(\infty) = 1$ for any value of the ratio m/n; a local search solver would have $F(\infty) = 0.5$ on a set of instances at the 4.3 threshold, $F(\infty) = 1$ below the threshold, and $F(t) = 0 \ \forall t$ above it.

In the following experiments, we used a set of 200 randomly generated instances at the phase transition, 100 of which were satisfiable, with $n = 250$ variables and $m = 1065$ clauses: the `uf-250-1065` and `uuf-250-1065` instances from SATLIB (Hoos and Stützle 2000). The algorithm portfolio consisted of two SAT solvers from the two categories above, the same as used by Gagliolo and Schmidhuber (2006b). As a complete solver we picked Satz-Rand (Gomes et al. 2000), a randomized version of Satz (Li and Anbulagan 1997) in which random noise influences the choice of the branching variable. Satz is a modified version of the complete DPLL procedure, in which the choice of the variable on which to branch next follows an heuristic ordering, based on first- and second-level unit propagation. Satz-Rand differs in that, after the list is formed, the next variable to branch on is randomly picked among the top h fraction of the list. We present results with the heuristic starting from the most constrained variables, as suggested also by Li and Anbulagan (1997), and the noise parameter set to 0.4. As a local search solver we used G2-WSAT (Li and Huang 2005): for this algorithm, we set a high noise parameter (0.5), as advisable for problems at the phase threshold, and the diversification probability at the default value of 0.05.

This algorithm set/problem set combination is quite interesting, as G2-WSAT almost always dominates the performance of Satz-Rand on satisfiable instances, while the latter is obviously the winner on all unsatisfiable ones, for which the runtime of G2-WSAT is infinite.

This situation is visualized by the continuous lines in Fig. 7.1, which plot the RTD of the set for the two solvers,[3] resulting from a KM estimate of the CDF (7.21)

[3] As we needed a common measure of time, and the CPU runtime measures are quite inaccurate (see also Hoos and Stützle 2004, p. 169), we modified the original code of the two algorithms by adding a counter that is incremented at every loop in the code. The resulting time measure was consistent with the number of backtracks, for Satz-Rand, and the number of flips, for G2-WSAT.

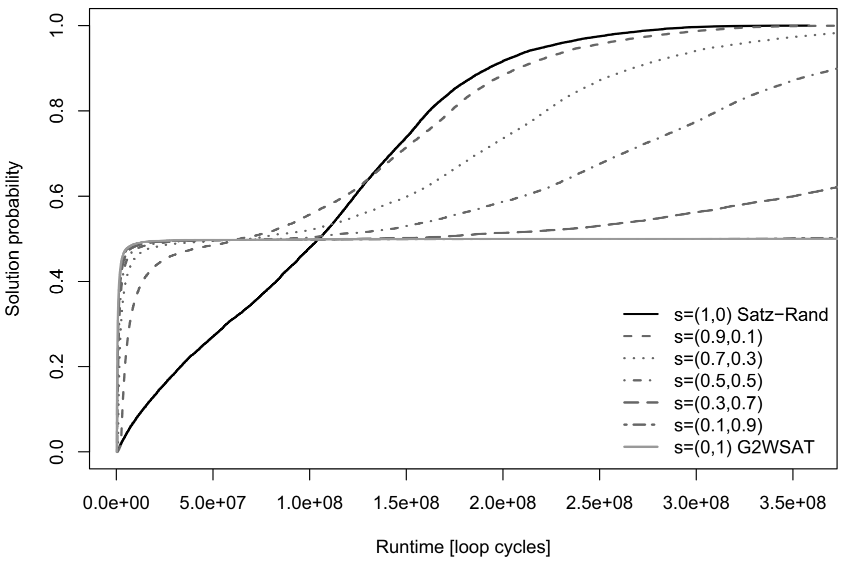
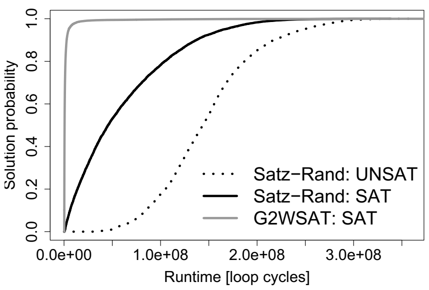


Fig. 7.1: The RTDs (continuous lines) of Satz-Rand (black) and G2WSAT (gray), along with the RTD of the portfolio (dashed lines) resulting from different values of the share $\mathrm{s} = (s_1, s_2)$, where s_1 is the portion of time allocated to Satz-Rand, and $s_2 = 1 - s_1$ is the portion used by G2WSAT

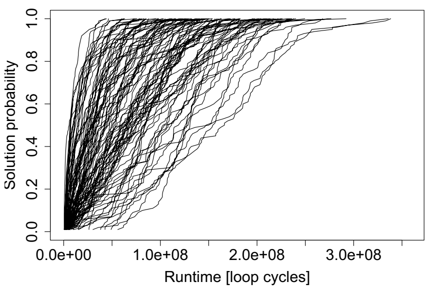
based on the runtimes collected from 100 runs with different random seeds for each of the 200 instances. No censoring was used, except of course for the runtimes of G2WSAT on the UNSAT instances, which were artificially generated as censored events with threshold 10^9. One can clearly notice the advantage of G2-WSAT on satisfiable instances, represented by the small lower quantiles (below 10^6). From quantile 0.5 on, the RTD remains flat, reflecting the fact that half of the instances are unsatisfiable. Satz-Rand starts solving problems later, and is competitive with G2-WSAT only on a small number of satisfiable instances, but is able to solve also all the unsatisfiable ones, as indicated by the fact that the RTD reaches 1.

On the same plot (dashed lines), we display the RTD of the portfolio (7.38) for different values of the share, obtained simulating a run of the portfolio (7.37) for each pair of runtime values. For $\mathrm{s} = (1, 0)$ only Satz-Rand is run. Giving a small portion of time to G2-WSAT ($\mathrm{s} = (0.9, 0.1)$) already improves the situation; increasing its share gradually moves the RTD of the portfolio towards the one of G2-WSAT ($\mathrm{s} = (0, 1)$).

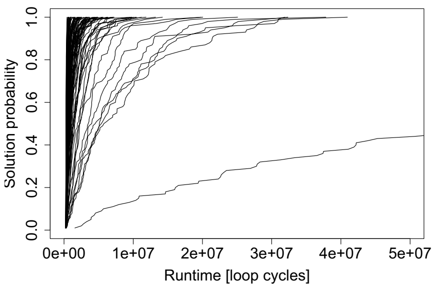
All runtimes reported for this benchmark are expressed in these loop cycles: on a 2.4 GHz machine, 10^9 cycles take about 1 minute.



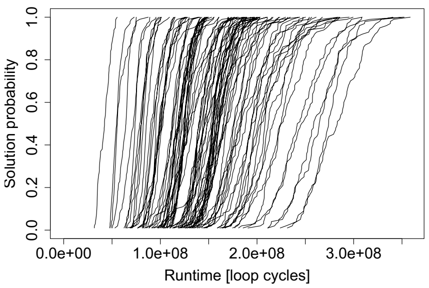
(a) RTD of SAT, UNSAT



(c) Satz-Rand on SAT instances



(b) G2WSAT on SAT instances

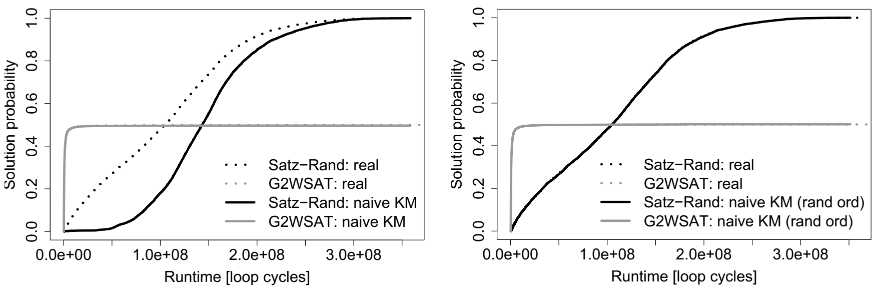


(d) Satz-Rand on UNSAT instances

Fig. 7.2: (a) RTDs of the two algorithms on the subsets of SAT and UNSAT instances: the line for G2WSAT on UNSAT instances would be constant at 0, and is omitted. (b) RTDs of G2WSAT on each of the 100 satisfiable instances. Note the different time scale. The lower line leaving the plot refers to instance 24, and reaches 1 at time 1.6×10^8. (c,d) RTDs of Satz-Rand on the satisfiable and unsatisfiable instances, respectively

Let us now look with more detail at the variability of the RTDs within the set. Figure 7.2(a) displays the RTDs of the two algorithms on the subsets of SAT and UNSAT instances. Figure 7.2(b,c,d) displays the RTDs of the instances, again estimated based on 100 runs for each instance, grouped based on the algorithm and on satisfiability. Note that there still is a huge variability of the RTD of the instances within each subset.

To illustrate the bias induced by a competing risk, in Fig. 7.3(a) we display again the unbiased RTDs of the two algorithms (dotted lines) on the whole set of instances, compared with the KM estimates (7.20) of the RTDs of the two algorithms, this time obtained with the portfolio approach, that is, censoring the runtime of the slowest algorithm for each run and each instance (continuous lines). While the model for G2WSAT remains accurate, as it mostly gets censored on unsatisfiable instances on which it would run forever anyway, one can clearly notice the bias of the product-limit estimator for Satz-Rand: the runtime is overestimated, especially for the satisfiable instances, on which Satz-Rand is slower, so it gets censored.



(a) Same instances (b) Random reordering

Fig. 7.3: (a) The unbiased RTDs (dotted lines) of Satz-Rand (black) and G2WSAT (gray), compared with the biased estimates (continuous lines) obtained by censoring, for each instance and each seed, the runtime of the slowest algorithm. Note the bias in the model for Satz-Rand. (b) The same unbiased RTDs (dotted lines), again compared with the estimates (continuous lines) obtained by censoring the runtime of the slowest algorithm, this time after a random reordering of the instances, in order to reduce the bias. Note the improvement in the model for Satz-Rand

To reduce the bias, we repeated the sampling, randomly reordering the instances for Satz-Rand: in this way, the two algorithms run in parallel, but each on a different instance. This should reduce the statistical dependence among the runtimes of the two algorithms, allowing to be considered the censoring mechanism uninformative, thus resulting in a more correct estimate: this can indeed be observed in Fig. 7.3(b), where we display again the "real" runtime distributions, compared against the estimates obtained after random reordering of the instances. This time the estimator for Satz-Rand is visibly more accurate on the whole range of runtimes observed.

7.5 Related Work

Literature on RTD modeling aimed at analyzing algorithm performance is relatively recent. The behavior of complete SAT solvers on solvable and unsolvable instances near phase transition have been shown to be approximable by Weibull and lognormal distributions, respectively (Frost et al. 1997). Heavy-tailed[4] behavior is observed for backtracking search on structured underconstrained problems by Hogg and Williams (1994), Gomes et al. (2000). The performance of local search SAT solvers is analyzed by Hoos (1999), and modeled using a mixture of exponential distributions by Hoos (2002).

[4] A *heavy-tailed* runtime distribution $F(t)$ is characterized by a Pareto tail, $F(t) \rightarrow_{t\rightarrow\infty} 1-Ct^{-\alpha}$. In practice, this means that most runs are relatively short, but the remaining few can take a very long time.

A seminal paper in the field of algorithm selection is that by Rice (1976), in which offline, per instance selection is first proposed, for both decision and optimization problems. More recently, similar concepts have been proposed, with different terminology (algorithm *recommendation*, *ranking*, *model selection*), in the *Meta-Learning* community (Vilalta and Drissi 2002). Research in this field usually deals with optimization problems, and is focused on maximizing solution quality, without taking into account the computational aspect.

Work on *Empirical Hardness Models* (Leyton-Brown et al. 2002, Nudelman et al. 2004) is instead applied to decision problems, and focuses on obtaining accurate models of runtime performance, conditioned on numerous features of the problem instances, as well as on parameters of the solvers. The models are used to perform algorithm selection on a per instance basis, and are learned offline: censored sampling is also considered (Xu et al. 2008).

The foundation papers about algorithm portfolios (Huberman et al. 1997, Gomes et al. 1998, Gomes and Selman 2001) describe how to evaluate the runtime distribution of a portfolio, based on the runtime distributions of the algorithms, which are assumed to be available. The RTD is used to evaluate mean and variance, and find the (per set optimal) *efficient frontier* of the portfolio, i.e., the subset of all possible allocations in which no element is dominated in both mean and variance.

Another approach based on runtime distributions is given by Finkelstein et al. (2002, 2003), for parallel independent processes and shared resources, respectively. The RTDs are assumed to be known, and the expected value of a cost function, accounting for both wall clock time and resources usage, is minimized. A dynamic schedule is evaluated offline, using a branch-and-bound algorithm to find the optimal one in a tree of possible schedules. Examples of allocation to two processes are presented with artificially generated runtimes, and a real Latin square solver. Unfortunately, the computational complexity of the tree search grows exponentially in the number of processes.

Other examples of the application of performance modeling to resource allocation are provided by the literature on restart strategies (Luby et al. 1993, Gomes et al. 1998), and on anytime algorithms (Boddy and Dean 1994, Hansen and Zilberstein 2001).

7.6 Summary and Outlook

In this chapter we illustrated the application of survival analysis methods to model the performance of decision problem solvers, focusing on the application of modeling to algorithm selection. We described in deeper detail the statistical aspects of GambleTA (Gagliolo and Schmidhuber 2006b, 2008a), discussing the bias in the RTD models caused by the *competing risks* censoring scheme adopted for sampling the runtime of the algorithms, and illustrating it with a simple sampling experiment.

Ongoing research is aimed at analyzing the actual impact of this bias on the performance of GambleTA, and at finding a computationally cheap way of reducing this bias, possibly inspired by the random reordering trick described in Sect. 7.4. In the longer term, we are working on transferring other survival analysis methods to model the runtime quality trade-off of optimization algorithms in order to devise an algorithm selection method for this broader class of problems.

Acknowledgements The authors would like to thank an anonymous reviewer, and Faustino Gomez, for useful comments on a draft of this chapter. The first author was supported by the Swiss National Science Foundation with a grant for prospective researchers (n. PBTI2–118573).

References

Aalen O (1978) Nonparametric inference for a family of counting processes. Annals of Statistics 6:701–726

Akritas M (1994) Nearest neighbor estimation of a bivariate distribution under random censoring. Annals of Statistics 22:1299–1327

Auer P, Cesa-Bianchi N, Freund Y, Schapire RE (2003) The nonstochastic multi-armed bandit problem. SIAM Journal on Computing 32(1):48–77

Beran R (1981) Nonparametric regression with randomly censored survival data. Tech. rep., University of California, Berkeley, CA

Bishop CM (1995) Neural networks for pattern recognition. Oxford University Press

Boddy M, Dean TL (1994) Deliberation scheduling for problem solving in time-constrained environments. Artificial Intelligence 67(2):245–285

Collet D (2003) Modeling survival data in medical research. Chapman & Hall/CRC, Boca Raton

Cox D (1972) Regression models and life-tables. Journal of the Royal Statistics Society, Series B 34:187–220

Cox D, Oakes D (1984) Analysis of survival data. Chapman & Hall, London

Finkelstein L, Markovitch S, Rivlin E (2002) Optimal schedules for parallelizing anytime algorithms: The case of independent processes. Tech. rep., CS Department, Technion, Haifa, Israel

Finkelstein L, Markovitch S, Rivlin E (2003) Optimal schedules for parallelizing anytime algorithms: The case of shared resources. Journal of Artificial Intelligence Research 19:73–138

Fitzmaurice G, Davidian M, Verbeke G, Molenberghs G (2008) Longitudinal Data Analysis. Chapman & Hall/CRC

Fleming T, Harrington D (1991) Counting processes and survival analysis. Wiley, New York, NY

Frost D, Rish I, Vila L (1997) Summarizing CSP hardness with continuous probability distributions. In: Kuipers B, et al. (eds) Fourteenth National Conference on Artificial Intelligence, pp 327–333

Gagliolo M, Schmidhuber J (2006a) Impact of censored sampling on the performance of restart strategies. In: Benhamou F (ed) Principles and Practice of Constraint Programming, Springer, pp 167–181

Gagliolo M, Schmidhuber J (2006b) Learning dynamic algorithm portfolios. Annals of Mathematics and Artificial Intelligence 47(3-4):295–328

Gagliolo M, Schmidhuber J (2007) Learning restart strategies. In: Veloso MM (ed) Twentieth International Joint Conference on Artificial Intelligence, vol 1, AAAI, pp 792–797

Gagliolo M, Schmidhuber J (2008a) Algorithm selection as a bandit problem with unbounded losses. Tech. Rep. IDSIA - 07 - 08, IDSIA

Gagliolo M, Schmidhuber J (2008b) Towards distributed algorithm portfolios. In: Corchado JM, et al. (eds) International Symposium on Distributed Computing and Artificial Intelligence (DCAI 2008), Springer, pp 634–643

Gent I, Walsh T (1999) The search for satisfaction. Tech. rep., Dept. of Computer Science, University of Strathclyde

Gomes CP, Selman B (2001) Algorithm portfolios. Artificial Intelligence 126(1-2):43–62

Gomes CP, Selman B, Kautz H (1998) Boosting combinatorial search through randomization. In: Mostow J, et al. (eds) Fifteenth National Conference on Artificial Intelligence, pp 431–437

Gomes CP, Selman B, Crato N, Kautz H (2000) Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. Journal of Automated Reasoning 24(1–2):67–100

Hansen EA, Zilberstein S (2001) Monitoring and control of anytime algorithms: A dynamic programming approach. Artificial Intelligence 126(1–2):139–157

Hogg T, Williams CP (1994) The hardest constraint problems: a double phase transition. Artificial Intelligence 69(1–2):359–377

Hoos HH (1999) On the run-time behaviour of stochastic local search algorithms for SAT. In: Hendler J, et al. (eds) Sixteenth National Conference on Artificial Intelligence, pp 661–666

Hoos HH (2002) A mixture-model for the behaviour of SLS algorithms for SAT. In: Hendler JA (ed) Eighteenth National Conference on Artificial Intelligence, pp 661–667

Hoos HH, Stützle T (2000) SATLIB: An online resource for research on SAT. In: Gent I, et al. (eds) SAT 2000 — Highlights of Satisfiability Research in the Year 2000, IOS, pp 283–292

Hoos HH, Stützle T (2004) Stochastic Local Search : Foundations & Applications. Morgan Kaufmann

Huberman BA, Lukose RM, Hogg T (1997) An economic approach to hard computational problems. Science 27:51–53

Ibrahim JG, Chen MH, Sinha D (2001) Bayesian Survival Analysis. Springer

Kaplan EL, Meyer P (1958) Nonparametric estimation from incomplete samples. Journal of the American Statistical Association 73:457–481

Keilegom IV, Akritas M, Veraverbeke N (2001) Estimation of the conditional distribution in regression with censored data: a comparative study. Computational Statistics and Data Analysis 35:487–500

Klein JP, Moeschberger ML (2003) Survival Analysis: Techniques for Censored and Truncated Data, 2nd edn. Springer

Leyton-Brown K, Nudelman E, Shoham Y (2002) Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In: Van Hentenryck P (ed) Principles and Practice of Constraint Programming, pp 91–100

Li CM, Anbulagan (1997) Heuristics based on unit propagation for satisfiability problems. In: Georgeff MP, et al. (eds) Fifteenth International Joint Conference on Artificial Intelligence, pp 366–371

Li CM, Huang W (2005) Diversification and determinism in local search for satisfiability. In: Bacchus F, et al. (eds) Theory and Applications of Satisfiability Testing, Springer, pp 158–172

Li G, Doss H (1995) An approach to nonparametric regression for life history data using local linear fitting. Annals of Statistics 23:787–823

Liang K, Self S, Bandeen-Roche K, Zeger S (1995) Some recent developments for regression analysis of multivariate failure time data. Lifetime Data Analysis 1:403–415

Luby M, Sinclair A, Zuckerman D (1993) Optimal speedup of las vegas algorithms. Information Processing Letters 47(4):173–180

Machin D, Cheung Y, Parmar M (2006) Survival Analysis. A Practical Approach. Wiley, UK, second edition

Mackay DC (2002) Information Theory, Inference and Learning Algorithms. Cambridge University Press

Mitchell D, Selman B, Levesque H (1992) Hard and easy distributions of SAT problems. In: Swartout W (ed) Tenth National Conference on Artificial Intelligence, pp 459–465

Nelson W (1972) Theory and applications of hazard plotting for censored failure data. Technometrics 14:945–965

Nelson W (1982) Applied Life Data Analysis. Wiley, New York, NY

Nielsen J, Linton O (1995) Kernel estimation in a nonparametric marker dependent hazard model. Annals of Statistics 23:1735–1748

Nudelman E, Brown KL, Hoos HH, Devkar A, Shoham Y (2004) Understanding random SAT: Beyond the clauses-to-variables ratio. In: Wallace M (ed) Principles and Practice of Constraint Programming, Springer, pp 438–452

Pintilie M (2006) Competing Risks. A practical perspective. Wiley, New York, NY

Putter H, Fiocco M, Geskus R (2006) Tutorial in biostatistics: Competing risks and multi-state models. Statistics in Medicine 26:2389–2430

Rice JR (1976) The algorithm selection problem. In: Rubinoff M, Yovits MC (eds) Advances in computers, vol 15, Academic, New York, pp 65–118

Spierdijk L (2005) Nonparametric conditional hazard rate estimation: a local linear approach. Tech. Rep. TW Memorandum, University of Twente

Tsang E (1993) Foundations of Constraint Satisfaction. Academic, London and San Diego

Tsiatis A (1975) A nonidentifiability aspect of the problem of competing risks. Proceedings of the National Academy of Sciences of the USA 72(1):20–22

Vilalta R, Drissi Y (2002) A perspective view and survey of meta-learning. Artificial Intelligence Review 18(2):77–95

Wang JL (2005) Smoothing hazard rate. In: Armitage P, et al. (eds) Encyclopedia of Biostatistics, 2nd Edition, vol 7, Wiley, pp 4986–4997

Wichert L, Wilke RA (2005) Application of a simple nonparametric conditional quantile function estimator in unemployment duration analysis. Tech. Rep. ZEW Discussion Paper No. 05-67, Centre for European Economic Research

Xu L, Hutter F, Hoos HH, Leyton-Brown K (2008) SATzilla: Portfolio-based algorithm selection for SAT. Journal of Artificial Intelligence Research 32 (2008) 565-606 32:565–606

Chapter 8
On Applications of Extreme Value Theory in Optimization

Jürg Hüsler

Abstract We present a statistical study of the distribution of the objective value of solutions (outcomes) obtained by stochastic optimizers, applied for continuous objective functions. We discuss the application of extreme value theory for the optimization procedures. A short review of the extreme value theory is presented to understand the investigations. In this chapter three optimization procedures are compared in this context: the random search and two evolution strategies. The outcomes of these optimizers applied to three objective functions are discussed in the context of extreme value theory and the performances of the procedures investigated, analytically and by simulations. In particular, we find that the estimated extreme value distributions and the fit to the outcomes characterize the performance of the optimizer in one single instance.

8.1 Introduction

Several possibilities exist to find the optimum point of a continuous objective function. We consider a stochastic optimizer which is an algorithm based on probabilistic procedures. We consider single-objective optimizers which produce one scalar outcome per optimization run, being the best objective value found within the steps of the run. The quality of an optimizer is a rather important issue, and is discussed by analyzing the local minima or maxima and the behavior of the solution of the optimizer based on a number of steps and the initial starting point.

Stochastic optimisers estimate the optimum value of the objective function using a particular random procedure. Hence the optimization outcomes are random and can be described by their distribution which allows to assess the performance of the optimizer. Typically, the usual statistical measures of an estimator can be applied,

Jürg Hüsler
Dept. Mathematical Statistics, University of Bern, Sidlerstr. 5, CH-3012 Bern, Switzerland
e-mail: juerg.huesler@stat.unibe.ch

as bias, variance or mean square error (cf. Fonseca and Fleming (1996)). It would be even better to know approximately the distribution of the estimators. In particular we are interested in an optimizer which is consistent, i.e., which approaches the optimum value as the number of internal steps increases to infinity. For such an optimizer, the lower or upper endpoint of the distribution will be the optimum value. Statistical methods can be applied to estimate the endpoint of interest. Several statistical parametric approaches are known. Since in most cases very little is known about the objective function, and hence about the distribution of the outcomes, such parametric methods cannot be applied.

In this chapter we will consider the minimization problem since the results are easily transformed to the maximization problem. We want to analyze the distribution of outcomes and also the best of the outcomes found after several executions (runs). We denote the sample of outcomes of k independent algorithm executions by $\{Y_i,\ i = 1, \ldots, k\}$. Each Y_i denotes the minima of the same number n of internal steps of the algorithm.

Since we consider minima, it is of main interest to investigate the distribution of Y_i. This can be related to the extreme value theory which investigates the distribution of the extreme values, as minima or maxima and smallest or largest order statistics. Therefore the extreme value theory is briefly reviewed in the next section, to discuss possible applications for optimizers.

Section 8.3 is devoted to random search strategy and the application of the classical extreme value theory. Section 8.4 contains some theoretical results which support the empirical study of Sect. 8.3. Section 8.5 contains experiments using the evolution strategy procedures, together with the application of another, more appropriate, method of extreme value theory for the evolution optimizers.

8.2 Extreme Value Theory

If the number of internal steps n is large, the distribution of the minima may be found by the extreme value theory (EVT), which is based on asymptotic results as $n \to \infty$. Minima and maxima are related by a simple sign change of the data. Hence, the outcomes of an optimizer for the maximal value of an objective function is modeled in the same way.

8.2.1 Extreme Value Theory for Minima

The classical limit theorem for extremes states that minima of iid (independent and identically distributed) random variables X_j should have an extreme value limiting distribution as the sample size n tends to ∞, after an appropriate linear normalization; see, e.g., Fisher and Tippett (1928), Leadbetter et al. (1983) or Falk et al. (2004). This means that the distribution of the minimum $Y = \min_{j \le n} X_j$, linearly normalized by $a_n(> 0)$ and b_n, converges in distribution

$$\Pr\{Y \leq a_n x + b_n\} = 1 - (1 - F(a_n x + b_n))^n \to G(x), \quad x \in \mathbb{R},$$

where G is a suitable distribution function and F is the underlying distribution of X_j. This holds under certain quite general conditions, in particular, if the underlying distribution F is continuous. The theory is usually formulated for maxima, but it is easily transformed for minima, as mentioned above, by $\min X_j = -\max(-X_j)$. The theory states that the limit distributions G satisfy the *min-stability* property. This property allows to derive all the possible distributions G. They are called the *extreme value distributions* for minima. A distribution G is called min-stable if for every k there exist $c_k(> 0)$ and d_k such that

$$\Pr\{\min_{i\leq k} U_i \leq c_k x + d_k\} = 1 - (1 - G(c_k x + d_k))^k = G(x),$$

where U_i are iid random variables with distribution G. For maxima, the stability property is called max-stability, i.e., $\Pr\{\max_{i\leq k} U_i \leq c_k x + d_k\} = G^k(c_k x + d_k) = G(x)$.

Let us mention the extreme value distributions for minima: for $\mu \in \mathbb{R}, \sigma > 0$

$$\begin{aligned}
\Lambda(x) &= 1 - \exp\left(-\exp\left(\tfrac{x-\mu}{\sigma}\right)\right) && \text{if } x \in \mathbb{R} && \text{(Gumbel)}\\
\Phi_\alpha(x) &= 1 - \exp\left(-\left(-\tfrac{x-\mu}{\sigma}\right)^{-\alpha}\right) && \text{if } x \leq \mu,\ \ \alpha > 0 && \text{(Fréchet)}\\
\Psi_\alpha(x) &= 1 - \exp\left(-\left(\tfrac{x-\mu}{\sigma}\right)^{-\alpha}\right) && \text{if } x \geq \mu,\ \ \alpha < 0 && \text{(Weibull)}
\end{aligned}$$

The extreme value distributions are typically combined into one class of the so-called *generalized extreme value distributions* (GEV), by using in addition the shape parameter $\gamma = 1/\alpha \in \mathbb{R}$. For minima, we have

$$G_\gamma(x) = 1 - \exp\left(-\left(1 - \gamma\frac{x-\mu}{\sigma}\right)^{-1/\gamma}\right) \quad \text{for} \quad 1 - \gamma\frac{x-\mu}{\sigma} > 0, \gamma \neq 0$$

If $\gamma \to 0$, then $G_\gamma(x) \to G_0(x)$, writing

$$G_0(x) = 1 - \exp\left(-\exp\left(\frac{x-\mu}{\sigma}\right)\right), \quad \text{for } x \in \mathbb{R}.$$

This class is normalized such that $G_\gamma(0) = 1 - e^{-1}$, where the possible finite endpoints depend now on γ.

Note that the Fréchet distribution ($\gamma > 0$) has an infinite left endpoint and its domain of attraction (the class of distributions for which the minimum has a limiting Fréchet distribution) only includes distributions with an infinite left endpoint. On the other hand the Weibull distribution has a finite left endpoint and its domain of attraction only includes distributions with a finite left endpoint. Moreover, in the Weibull family, the shape parameter α determines the rate of growth of the density towards the endpoint: if $\alpha = 1/\gamma < -1$ the density decreases towards zero as it approaches

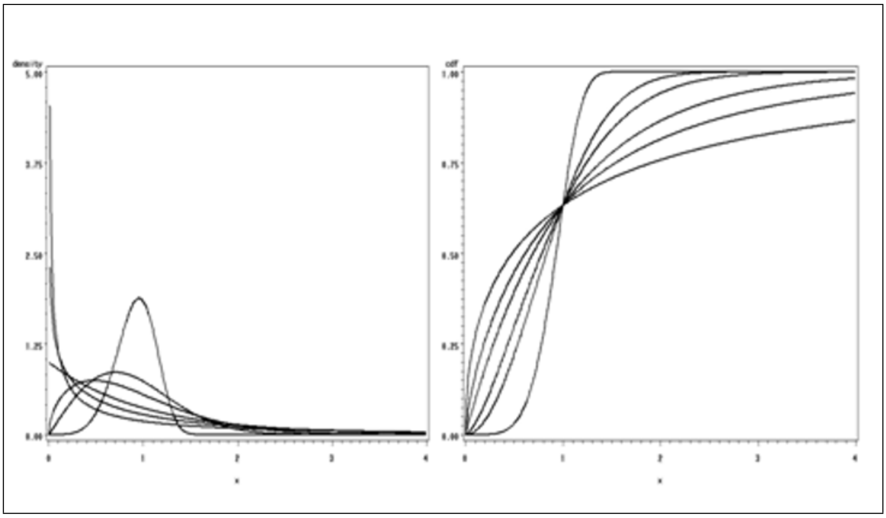


Fig. 8.1: Weibull densities and cumulative distributions functions (cdf) for several α's, $-\alpha = 0.5, 0.75, 1.0, 1.5, 2.0, 5.0$, from left to right, with $\mu = 0$ and $\sigma = 1$

the endpoint, whereas if $-1 < \alpha = 1/\gamma < 0$ the density increases monotonically towards infinity. Figure 8.1 illustrates this behavior. Finally the Gumbel distribution is between the Fréchet and Weibull distributions, having a rather heterogeneous domain of attraction including distributions with either a finite or an infinite endpoint. Its density looks like a Weibull density with a very negative shape parameter, or equivalently as a Fréchet density with a very large shape parameter since, as mentioned, the Gumbel distribution is the limit as $\gamma = 1/\alpha \to 0$.

We think that the extreme value distributions can be applied for optimizers if the assumptions of the extreme value result hold. This is true only for certain optimizers, such as for instance the random search one. In such a situation, Y_i is the minimum of n independent and identically distributed values of the algorithm. Since n is large, as typically is the case for the random search algorithm, we expect that the Y_i are approximately distributed following an extreme value distribution. This will be observed in Sect. 8.3 and derived analytically in Sect. 8.4. For finite large n, we expect that the distribution of the Y_i will be close to the limiting extreme value distribution. Hence it is reasonable to fit the distribution of the Y_i by an extreme value distribution and to estimate the parameters of the fitted extreme value distribution.

Several different estimates are known based on different estimation procedures, as maximum-likelihood estimation, moment estimation, Bayesian estimation, probability weighted moment estimation, least squares estimation, best linear unbiased or invariant estimation, and many more; see, e.g., the recent textbooks by Reiss and Thomas (1997), Beirlant et al. (2004), de Haan and Ferreira (2006), and Resnick (2007). We will apply maximum-likelihood estimators in the following examples, if appropriate. The goodness-of-fit can be assessed, which we will investigate in Sect. 8.3 in our simulation examples.

8.2.2 Peaks Over Threshold Method (POT) for Minima

When the number of internal steps n is possibly not large, or when these steps are dependent, the classical extreme value theory cannot be applied for optimizers. However, the distribution of $M_k = \min_{i \leq k} Y_i$ could still be approximated by an extreme value distribution if k is large, since the Y_i are iid random variables (r.v.'s). In addition, if the sample size k is large, then the smallest order statistics of the Y_i contain a good amount of information on the tail of the distribution of interest. If we consider the (negative) excesses below the m-th smallest value of the Y_i's, the conditional distribution of the excesses can be modeled approximately by a *generalized Pareto distribution*. This is the second possible approach of extreme value theory which intends in general to estimate the tail of a distribution. This part of EVT is called the *peaks-over-threshold (POT) method* with respect to maxima. In our context it would be the approach based on negative excesses below a low threshold. However, we keep this expression and also the term exceedances for an outcome above the threshold, hence for a negative excess in the context of minima.

Mostly one wants to estimate the extreme quantiles or the lower endpoint, as in the case of optimization. Under the same assumptions as for the limit distribution of minima, the generalized Pareto distributions are the only possible limits of the conditional distribution

$$F_{Y,t}(x) = \Pr\{Y < x | Y < t\} \quad \text{for } x < t, \quad \text{as } \ t \downarrow x_0,$$

where x_0 denotes the lower endpoint of F_Y and of $F_{Y,t}$.

The generalized Pareto distributions (for the lower tail) are defined by the shape parameter $\gamma \in \mathbb{R}$, the location parameter μ, and the scale parameter $\sigma > 0$:

$$W_\gamma(x) = \left(1 - \gamma \frac{x-\mu}{\sigma}\right)^{-1/\gamma} \quad \text{for} \quad \begin{cases} (x-\mu)/\sigma < 0 & \text{if } \ \gamma > 0 \\ 1/\gamma < (x-\mu)/\sigma < 0 & \text{if } \ \gamma < 0 \end{cases}$$

and

$$W_0(x) = \exp(x) \quad \text{for } \ x < 0, \ \text{if } \ \gamma = 0.$$

Again $W_\gamma(x) \to W_0(x)$ as $\gamma \to 0$.

The lower tail of the extreme value distribution G_γ for minima is asymptotically approximated by the generalized Pareto distribution W_γ: $G_\gamma(x) \sim W_\gamma(x)$ for $x \to x_0$. Hence, for the optimizer problem with finite global minimum, only Pareto distributions W_γ with $\gamma < 0$ are of interest. These distributions belong to the Beta family with parameter $\alpha = 1/\gamma$, as the Weibull distributions Ψ_α in relation to the extreme value distributions G_γ. We have $W^*_\alpha(x) = [(x-\mu)/\sigma]^{-\alpha}$ for

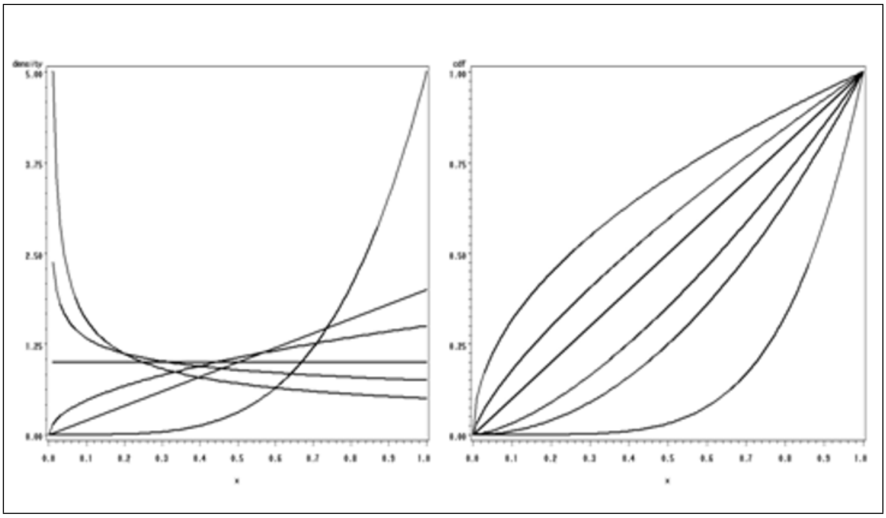


Fig. 8.2: Standard Beta densities and cumulative distributions functions $W_\alpha^*(x)$ for several α's, $-\alpha = 0.5, 0.75, 1.0, 1.5, 2.0, 5.0$, from left to right, with $\mu = 0$, $\sigma = 1$

$0 \leq (x-\mu)/\sigma \leq 1$ with $\alpha < 0$ (see Fig. 8.2). Since the location parameter μ plays the role of the lower endpoint of the Weibull or the Beta distributions in the case of $\alpha < 0$, we use μ instead of x_0 in the following. The parameter μ is the minimum of the objective function to be estimated with an optimizer.

Having computed k independent values Y_i with some optimizers, the lower tail and lower endpoint can be estimated with the use of the m smallest values of the Y_i's (with some $m < k$); see, e.g., Falk et al. (2004), Reiss and Thomas (1997), and de Haan and Ferreira (2006). Several procedures are also known for the estimation of the parameters of the generalized Pareto distributions and their quantiles. Some are applicable only for some γ, e.g., the Hill estimates for $\gamma > 0$. We can apply maximum-likelihood (ml) estimations (for $\gamma > -0.5$, sometimes even for $\gamma > -1$), the generalized Hill estimator, the moment estimator of Dekkers-Einmahl-de-Haan, the Drees-Pickands estimators, the moment estimators, the mixed moment estimator, kernel estimates, and many more; see e.g., Reiss and Thomas (1997), Beirlant et al. (2004), de Haan and Ferreira (2006), and Resnick (2007). For estimation of confidence intervals we can apply asymptotic formulae or the profile likelihood method.

Typically m is much smaller than k, to derive good results. If m is too small, the estimates are not reliable, but if m is too large, then the estimates are biased since information from an intermediate domain of the distribution is used and not from the lower tail only. So selection of m is a critical point. Several approaches for selection and estimation are proposed, which we discuss in the following applications.

We have no analytical results on the behavior of the smallest observations of an optimizer which has dependent internal steps besides of the given argument on the tail of the distribution. We do not know which generalized Pareto distribution or Beta distribution should be fitted to the outcomes of a particular optimizer in relation to an objective function. However, the simulation results presented in Sect. 8.5 support our suggestion of applying the POT approach.

8.2.3 Assessment of an Optimizer

If an optimizer produces accurate solutions, the outcomes should cluster near the optimum. Thus the limiting distribution of M_k or of Y_i (if n is large) is expected to be Weibull. Moreover, the greater the clustering effect is, the smaller the absolute value of the shape parameter $\alpha = 1/\gamma$ is and the better the solutions are. The shape parameter of the limiting distribution of M_k may depend also on the number n of internal steps.

If solutions are far from the optimum due to insufficient number of steps, either a Weibull fit with a very negative α is expected, or even a Fréchet or Gumbel fit could result, clearly indicating that the optimum is far away (possibly estimated at infinity).

We perform a set of simulation studies in Sect. 8.3 with the random search optimizer in order to investigate the use of extreme value modeling in optimization problems. We study the relation between the parameters of the distribution and the performance of the optimizer. As mentioned, some performances are supported by analytical results (Sect. 8.4).

For each simulation situation considered, the parameters of the extreme value distribution which fits the outcomes of the optimizer must be estimated. The estimation of the extreme value parameters may be done with different procedures, as mentioned above; for details see Falk et al. (2004), Embrechts et al. (1997) or de Haan and Ferreira (2006). In the approach based on EVT, the whole sample is used for estimation by assuming that the data are minima values and have an extreme value distribution. In the approach based on POT, it is assumed that the whole sample may not follow an extreme value distribution and only the lower order statistics of the sample are used for estimation since the limiting extreme value distribution characteristics are only contained in the left tail of the distribution of the original variables Y_i.

We applied the EVT approach approach with maximum-likelihood (ml) estimators as implemented in *Xtremes* (software package included in Reiss and Thomas (1997)). This package allows data to be modeled with the EVT and POT approach, with different estimation procedures, goodness-of-fit-tests, and graphical devices to judge the modeling. Also many of the EVT procedures exist as routines in the software language R. Note that the modeling in *Xtremes* and R is written for maxima, so minima have to be transformed first by sign change, as mentioned.

For those situations where ml estimation fails (typically when $-2 < \alpha < 0$) we used an alternative method suggested by Smith (1986). In this case the endpoint μ is estimated by the sample minimum and the other two parameters by ml. We could also apply the moment estimator by Dekkers et al. for the POT approach with negative γ, see Dietrich et al. (2002), Reiss and Thomas (1997) or de Haan and Ferreira (2006). This consistent estimator is defined for the lower tail as

$$\hat{\gamma} = m_{1,m} + 1 - \frac{1}{2(1 - m_{1,m}^2/m_{2,m})},$$

where $m_{j,m} = \sum_{i=0}^{m-1}(\log(y_{(i)}/y_{(m)}))^j/m$, for $j = 1, 2$, with the ordered data $y_{(i)}$.

In order to assess the goodness of fit of the Weibull distribution in the EVT approach we use the A^2 test statistic described by Lockhart and Stephens (2006) developed for the three parametric Weibull distributions $\Psi_\alpha(x; \mu, \sigma)$ where the three parameters are unknown. The three parameters location μ, scale σ, and shape α are estimated by maximum-likelihood estimation, resulting in $\hat{\mu}$, $\hat{\sigma}$, and $\hat{\alpha}$. Then for the ordered data set $y_{(i)}, 1 \le i \le k$, let $z_{(i)} = \Psi_\alpha(y_{(i)}; \hat{\mu}, \hat{\sigma})$ and define the goodness-of-fit test statistic

$$A^2 = -k - \frac{1}{k}\sum_{i=1}^{k}(2i-1)[\log z_{(i)} + \log(1 - z_{(k-i+1)})].$$

The critical values of this test statistic were derived by Lockhart and Stephens (2006), based on the asymptotic distribution of A^2, and tabulated for applications. Monte Carlo simulations showed that the critical values can be used even for smaller samples of size $k \ge 10$. This test should not be used to test the goodness-of-fit in the POT approach.

8.3 Experiments Using Random Search

Random search has been chosen in our investigation since its execution requires very little time, the algorithm is very simple and can be applied for many different experiments, and it serves as motivation for further research.

All simulation studies presented by Hüsler et al. (2003) were performed using Maple, Matlab, and Xtremes. An interface platform was developed to transfer data automatically between the different software packages used.

In this section we deal with the random search algorithm only. In Sect. 5 we will investigate two evolutionary algorithms. Several objective functions $g : \mathbb{R}^2 \to \mathbb{R}$, are considered:

- Rosenbrock function $(1-x)^2 + 100(y - x^2)^2$
- Simple quadratic function $ax^2 + by^2,\ a, b \in \mathbb{R}^+$
- A function with local minima and multiple global minima $\sin(x^2 + y^2)/(x^2 + y^2)$ denoted by *sinc*.

The minimum values of the Rosenbrock and the quadratic functions are 0 at $(x, y) =$

$(1, 1)$ and $(x, y) = (0, 0)$, respectively, whereas the minimum of the sinc function is -0.2172 at $x^2 + y^2 = 4.4935$.

The analytic behavior of g in the neighborhood of the minimum value is discussed in Sect. 8.4, showing how the underlying distribution of the random search algorithm and the behavior of g influence the behavior of the outcomes.

The random search algorithm randomly generates a prefixed (large) number n of iid random vectors (for our examples in $\mathbb{R}^2$) during an execution run and the value of the objective function is calculated for each one. The outcome will then be the minimum value of these values. In our investigation, the random vectors were generated by the bivariate normal distribution with mean $\xi \in \mathbb{R}^2$ and diagonal covariance matrix $\delta^2 I$ (where I is the 2×2 identity matrix).

By using different values of parameters ξ and δ we may illustrate and interpret different optimizer behaviors. All simulations are based on the same large number of internal steps $n = 600$ and repetitions of the independent algorithm executions with sample size $k = 500$. The latter was chosen to reduce sufficiently the simulation error in the statistical estimation. The behavior of the optimizer may depend on the starting values, here on the selection of the parameters ξ and δ. In particular, we may choose ξ at the optimum or near the optimum or further away. This behavior is investigated in the following paragraphs.

8.3.1 Samples with Simulations Near the Optimum

In order to assure that solutions are near the optimum, random search is performed mostly with ξ equal to the optimum. If ξ is not the optimum, which is the typical situation, then δ is chosen large enough to guarantee that the optimum is approached. Various δ values are chosen to investigate the possibly different behavior. Table 8.1 contains a set of representative cases. It reports for each case the estimation of the parameters of an extreme value distribution fitting the sample Y_i. The last column refers to the goodness-of-fit test A^2 for the Weibull distribution.

Since each outcome is the smallest value among $n = 600$ simulated iid random observations, it is expected that the distribution of the outcomes is itself close to an extreme value distribution. Looking at the p-values of the A^2 test listed in the table, we see that some are rather critical and suggest a poor fit. Figures 8.3 and 8.4 contain QQ-plots for cases 3 and 4 (poor and good fit, respectively). In Fig. 8.3 we observe one rather large value which influence the p-value. This value is not the starting point of the simulation. It is a result of the rather special behavior of the Rosenbrock function. These examples are discussed further in the analytical Sect. 8.4, where the analytic minimum points and minimum value are mentioned.

Table 8.1: Solutions around the optimum (minimum) for several cases with sample size $k = 500$, with p-values of the A^2 test. The minimum value of the objective function is estimated by $\hat{\mu}$

Case	function	ξ	δ	$\hat{\alpha}$	$\hat{\mu}$	$\hat{\sigma}$	p-value
1	Rosenbrock	optimum (1,1)	0.01	−1.01	3.69e−11	3.34e−6	0.037
2	Rosenbrock	optimum (1,1)	0.50	−0.98	1.78e−07	0.0082	0.110
3	Rosenbrock	optimum (1,1)	1.00	−0.99	1.61e−05	0.034	0.011
4	Rosenbrock	(0,0)	2.00	−1.00	4.52e−04	0.172	0.500
5	$0.001(x^2 + y^2)$	optimum (0,0)	1.00	−1.05	9.03e−09	3.36e−6	0.392
6	$(1000x^2 + y^2)$	optimum (0,0)	1.00	−0.96	5.04−05	0.103	0.457
7	*sinc*	$(0, 0)$	2.00	−0.49	−0.2165	1.61e−05	0.044

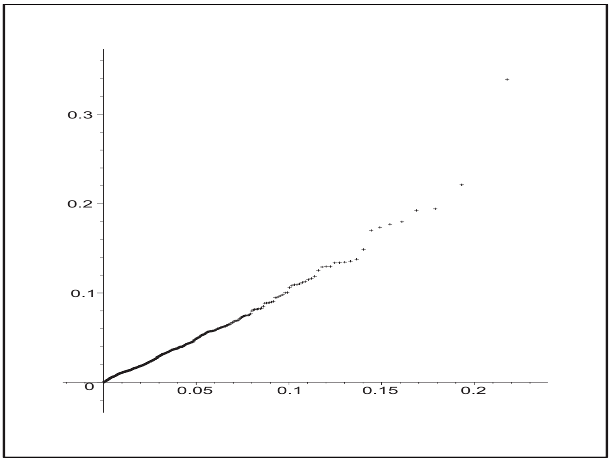


Fig. 8.3: QQ-plot of random search data of case 3 in Table 8.1, A^2-test: p-value = 0.011 (x-coordinate: Weibull quantiles, y-coordinate: sample quantiles)

8.3.2 Samples with Simulations Away from the Optimum

Table 8.2 contains a set of representative situations where random search is generated away from the optimum which happens if the center ξ of simulations is away from the optimum and also the variance component δ is small.

All cases reported in Table 8.2 clearly ended in a transition state ($\hat{\mu}$ is still quite far from the true minimum) and the estimated $\hat{\alpha}$ values are very large in magnitude. The results are much better if δ is selected appropriately large.

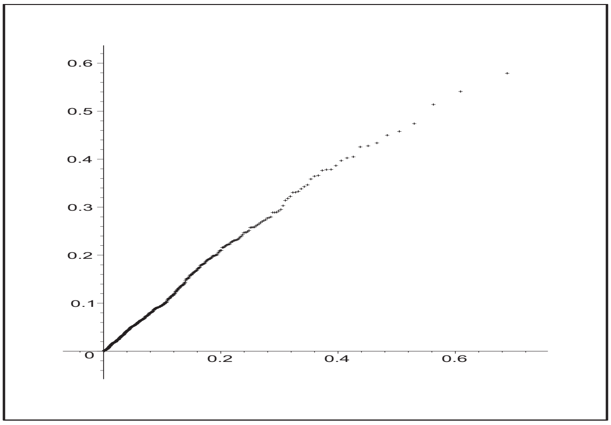


Fig. 8.4: QQ-plot of random search data of case 6 in Table 8.1, A^2-test: p-value = 0.457 (x-coordinate: Weibull quantiles, y-coordinate: sample quantiles)

Table 8.2: Solutions away from the optimum (minimum) for several cases with sample size $k = 500$, showing transition state solutions, with p-values of the A^2 test. The minimum value of the objective function is estimated by $\hat{\mu}$

Case	Function	ξ	δ	$\hat{\alpha}$	$\hat{\mu}$	$\hat{\sigma}$	p-value
1	Rosenbrock	$(0,0)$	0.01	-5.94	0.908	0.0389	0.50
2	$0.001(x^2+y^2)$	(3,3)	0.10	-5.76	0.014	0.0014	0.06
3	$1000x^2 + y^2$	(3,3)	0.10	-9.42	5712.2	1612.1	0.05

8.4 Analytical Results

From the experiments described in Sect. 8.3.1 it is clear that, excluding the *sinc* case (7) in Table 8.1, the estimated shape parameter α is close to -1 whenever the solutions are near the optimum and very negative when far from the optimum. For the *sinc* case, $\hat{\alpha}$ is close to -0.5. Also the $\hat{\alpha}$ values do not seem to be affected by a change of δ (scale) unless this change modifies the type of solution found.

We mention some analytical results and examples given by Hüsler et al. (2003), which clearly support the empirical results described above for the random search algorithm, letting n, the number of steps in an optimization run, grow to infinity.

We consider the general situation with an objective function $g : \mathbb{R}^d \to \mathbb{R}$. Let $\boldsymbol{X}_j$, $j = 1, \ldots, n$, be iid random vectors in $\mathbb{R}^d$ with $\boldsymbol{X}_j \sim F$ and density f. These vectors will denote the points generated within an optimization run, which for random search are the steps of the run. Assume that the objective function g has a global minimum $g_{\min}$. If the set of global minimum points is countable we denote the global minimum points by $\boldsymbol{x}_l, 1 \leq l \leq L$, where L is finite or infinite.

Define the domain $A_u = \{\boldsymbol{x} \in \mathbb{R}^d : 0 \leq g(\boldsymbol{x}) - g_{\min} \leq u\}$ for u small. The domain A_u can be bounded or unbounded depending on the function g. If the domain is bounded, we consider its d-dimensional volume $|A_u|$. Furthermore, if A_u is concentrated in a lower-dimensional subspace $\mathbb{R}^r$ of $\mathbb{R}^d$, then $|A_u| = 0$. It is more reasonable to define the r-dimensional volume $|A_u|_r$ and consider also the marginal density f_r of f in this lower-dimensional space related to A_u, $r \leq d$. We consider the limiting behavior of the distribution of the outcomes, Y_i being the minimum of n outcomes $g(X_j)$ within one run, when $n \to \infty$.

With the independence of the $\boldsymbol{X}_j$ within each run, we get for $Y_1 = \min_{j \leq n} g(\boldsymbol{X}_j)$

$$\begin{aligned}
\Pr\{Y_1 > g_{\min} + u\} &= \Pr\{\min_{j \leq n} g(\boldsymbol{X}_j) > g_{\min} + u\} \\
&= \Pr\{g(\boldsymbol{X}_j) > g_{\min} + u,\ j \leq n\} \\
&= (1 - \Pr\{g(\boldsymbol{X}_1) \leq g_{\min} + u\})^n \\
&= (1 - \Pr\{\boldsymbol{X}_1 \in A_u\})^n = (1 - p(u))^n, \qquad (8.1)
\end{aligned}$$

where $p(u) = \Pr\{\boldsymbol{X}_1 \in A_u\}$. We note that the asymptotic behavior of the minimum $\min_j g(\boldsymbol{X}_j)$ depends on the domain A_u or more precisely on the probability $p(u)$ that $\boldsymbol{X}$ hits A_u. This probability $p(u)$ tends usually to 0 as $u \to 0$, e.g., if the set of global minimum points is countable. The case with $p(u) \to p > 0$ is less interesting for the study of optimizers. If we can normalize this probability $p(u_n)$ with u_n such that $n\,p(u_n) \to \tau$ for some sequence u_n, then we get immediately that $(1 - p(u_n))^n \to \exp(-\tau)$ as $n \to \infty$. Hence, we may derive limit laws, e.g., for the linearly normalized minimum $(\min_{j \leq n} g(\boldsymbol{X}_j) - g_{\min})/a_n$ for some normalization constants $a_n > 0$. The following Theorem 8.1 is a general result which follows immediately from (8.1) and the assumed condition on the normalization u_n. This result explains the behavior of the random search optimizer in our examples of objective functions.

Theorem 8.1. *Assume that g has a global minimum value $g_{\min}$. Assume that A_u and the iid random vectors $\boldsymbol{X}_j$ ($j \geq 1$) are such that $p(u) = \Pr\{\boldsymbol{X}_j \in A_u\} \to 0$ as $u \to 0$. If there exists a normalization $u = u_n(x) \to 0$ (as $n \to \infty$) such that $np(u_n) \to h(x)$, for some $x \in \mathbb{R}$, then as $n \to \infty$*

$$\Pr\{Y_1 = \min_{j \leq n} g(\boldsymbol{X}_j) \leq g_{\min} + u_n(x)\} \to 1 - \exp(-h(x)).$$

If the function g has some isolated global minimum points $\boldsymbol{x}_l, l \leq L$, we can derive a more explicit statement. Assume that the set A_u can be split into disjoint sets $A_u(\boldsymbol{x}_l) = \{\boldsymbol{x} : g(\boldsymbol{x}) - g_{\min} \leq u \text{ and } |\boldsymbol{x} - \boldsymbol{x}_l| < \epsilon\}$ for some $\epsilon > 0$ and all sufficiently small u. The choice of ϵ has no impact; it is only necessary that the sets $A_u(\boldsymbol{x}_l)$ are disjoint for all u sufficiently small. Such cases will be discussed in the examples below.

Theorem 8.2. *Assume that g has a countable number of isolated global minimum points $\boldsymbol{x}_l, 1 \leq l \leq L \leq \infty$. Assume that each of the disjoint sets $A_u(\boldsymbol{x}_l)$ is bounded* *and concentrated in $I\!R^r$ with $r \leq d$, for all small u and every $l \leq L$, and that the random vector $\boldsymbol{X}$ has a positive, uniformly continuous (marginal) density f_r at the global minimum points $\boldsymbol{x}_l, l \leq L$, where the marginal density f_r corresponds to the space of A_u. If u_n is such that for $l \leq L$ uniformly*

$$n|A_{u_n}(\boldsymbol{x}_l)|_r \to \tau_l < \infty \quad \textit{with} \quad \sum_{l \leq L} f_r(\boldsymbol{x}_l)\tau_l < \infty,$$

then as $n \to \infty$

$$\Pr\{Y_1 = \min_{j \leq n} g(\boldsymbol{X}_j) \leq g_{\min} + u_n\} \to 1 - \exp\left(- \sum_{l \leq L} f_r(\boldsymbol{x}_l)\tau_l \right).$$

Proof. The proof is straightforward using (8.1) and the assumptions on the disjoint sets and the continuity of f_r to approximate $\Pr\{\boldsymbol{X}_1 \in A_u(\boldsymbol{x}_l)\}) \sim f_r(\boldsymbol{x}_l)|A_{u_n}(\boldsymbol{x}_l)|_r$. For more details see the proof by Hüsler et al. (2003).

Example 1: Assume that $g(x,y) = ax^2 + by^2$ for some $a, b > 0$. Then the miminum of g is at the origin $\mathbf{0} = (0,0)$ and the bounded domain $A_u = \{(x,y) : ax^2 + by^2 \leq u\}$ is an ellipsoid in $\mathbb{R}^2$ for $u > 0$. The volume of the ellipsoid A_u is $|A_u| = cu$ with constant $c = \pi/\sqrt{ab}$. Hence, select $u_n = x/(cn)$, $x > 0$. Let $\boldsymbol{X}$ be any bivariate random vector with continuous density f at the origin $\mathbf{0}$ with $f(\mathbf{0}) > 0$. Then by Theorem 8.2 as $n \to \infty$

$$\Pr\{Y_1 = \min_{j \leq n} g(\boldsymbol{X}_j) \leq g_{\min} + x/(cn)\} \to 1 - \exp(-f(\mathbf{0})x)$$

for $x > 0$. Thus $\alpha = -1$ for the Weibull limit distribution. This example can be extended simply to one with a finite number of global minimum points. The function $g(x,y) = a(x-c_1)^2(x-c_2)^2 + by^2$ has two global minimum points $(c_1, 0)$ and $(c_2, 0)$. Similarly, we can generalize further to have four global minimum points $(c_k, d_l), k, l = 1, 2$ by letting $g(x,y) = a(x-c_1)^2(x-c_2)^2 + b(y-d_1)^2(y-d_2)^2$. The result for $\min_{j \leq n} g(\boldsymbol{X}_j)$ is obvious in these cases.

Example 2: We consider now the two-dimensional *sinc* function $g(x,y) = \sin(x^2+y^2)/(x^2+y^2)$. The minimum is attained at points (x,y) with $x^2+y^2 = r_0^2$, where r_0 is the smallest positive solution of $r_0^2 \cos r_0^2 = \sin r_0^2$, i.e., $r_0 = 2.1198$. Therefore $g_{\min} = \cos r_0^2$. Hence the domain A_u is a ring with center $\mathbf{0}$ and radii $r_0 \pm \sqrt{2u/c}$ for some constant $c = \tilde{g}(r_0) + o(1)$, where $\tilde{g}(r) = (\sin r^2)/r^2$. Then we need to derive $\Pr\{\boldsymbol{X} \in A_u\}$ for $u \to 0$, where $\boldsymbol{X}$ is bivariate normal with center (ξ_1, ξ_2) and covariance matrix $\delta^2 I$. This is approximately the width of the ring times the integral of the density on the circle with radius r_0:

$$\Pr\{\boldsymbol{X} \in A_u\} \sim 2\sqrt{2u/c} \int_0^{2\pi} \frac{r_0}{2\pi\delta^2} \exp\left(-\frac{(r_0 \cos\phi - \xi_1)^2 + (r_0 \sin\phi - \xi_2)^2}{2\delta^2} \right) d\phi$$
$$= Du^{1/2}.$$

Select now $u_n(x) = x/(Dn)^2$ for $x > 0$, to get as $n \to \infty$

$$\Pr\{Y_1 = \min_{j \le n} g(\boldsymbol{X}_j) \le \cos r_0^2 + x/(Dn)^2\} \to 1 - \exp(-x^{1/2}) .$$

Hence $\alpha = -0.5$ for the Weibull limit distribution which we observed in the simulation results (see Table 8.2).

Example 3: Let us now choose $g(x, y) = x^2$. The set A_u in $\mathbb{R}^2$ is unbounded: $A_u = \{(x, y) : |x| \le \sqrt{u}\}$. But only the first component is relevant for the minimum. If the random vector $\boldsymbol{X}$ is standard bivariate Gaussian with correlation $\rho \in (-1, 1)$, then $\Pr\{\boldsymbol{X} \in A_u\} = \Pr\{|X_1| \le \sqrt{u}\} = 2\Phi(\sqrt{u}) - 1$ where Φ denotes the unit normal cumulative distribution function. Thus $\Pr\{\boldsymbol{X} \in A_u\} \sim 2\sqrt{u/2\pi}$ as $u \to 0$. Hence, select $u_n = \pi x/(2n^2)$ to get by Theorem 8.1

$$\Pr\{Y_1 = \min_{j \le n} g(\boldsymbol{X}_j) \le \pi x/(2n^2)\} \to 1 - \exp(-x^{1/2})$$

for $x > 0$ and $\alpha = -0.5$ of the Weibull limit distribution. This shows a case which is not directly included in Theorem 8.2. But if we consider $A_u \in \mathbb{R}$ being the interval $(-\sqrt{u}, \sqrt{u})$ since g is only a function of x, then we can apply the result of Theorem 8.2 with the 1-dimensional volume, the length of $A_u = 2\sqrt{u}$, and the corresponding marginal density $f_1(0) = 1/\sqrt{2\pi}$.

If the function g is rather regular, i.e., twice continuously differentiable at a unique global minimum point, we can state another general result. If there are finitely many isolated global minimum points, it is obvious how to extend the following result.

Theorem 8.3. *Assume that g is twice continuously differentiable at the unique global minimum point, say at the origin, and that the random vector $\boldsymbol{X}$ has a positive continuous (marginal) density f at the origin. If the Hessian matrix H has full rank d and $u_n(x) = n^{-2/d}x$, then as $n \to \infty$*

$$\Pr\{Y_1 = \min_{j \le n} g(\boldsymbol{X}_j) \le g_{\min} + n^{-2/d}x\} \to 1 - \exp\left(-x^{d/2} f(\mathbf{0}) c_d \prod_{l \le d} \lambda_l^{-1/2}\right) ,$$

for any $x > 0$, with $c_d = \pi^m/m!$ for $d = 2m$ even, and $c_d = 2^{m+1}\pi^m/(1 \cdot 3 \cdot 5 \cdot \ldots \cdot d)$ for $d = 2m+1$ odd, the volume of the d-dimensional unit sphere, and λ_l the positive eigenvalues of H.

Proof. By assumptions $g(\boldsymbol{x}) = g_{\min} + \boldsymbol{x}^T H \boldsymbol{x} + o(|\boldsymbol{x}|^2)$. Hence A_u is (approximately) an ellipsoid with a finite $|A_u|$ for all sufficiently small u, and we can approximate

$$\Pr\{X \in A_{u_n}\} \sim f(\mathbf{0})|A_{u_n}| \sim f(\mathbf{0}) c_d \prod_{l \le d} \lambda_l^{-1/2} u_n^{d/2}$$

by the continuity of the density f at $\mathbf{0}$, where $c_d \prod_{l \le d} \lambda_l^{-1/2}$ is the volume of the ellipsoid in $\mathbb{R}^d$ with axes $\lambda_l^{-1/2}$. Thus the statement follows by Theorem 8.1 and the normalization.

Example 4: We consider the generalized Rosenbrock function $g(x, y) = (1 - x)^2 + a(y - x^2)^2$ for some $a > 0$. In the simulation we used the standard constant $a = 100$. The function g has continuous second derivatives; the unique global minimum point is at $\mathbf{1} = (1, 1)$. The Hessian matrix is

$$H = \begin{pmatrix} 8a + 2 & -4a \\ -4a & 2a \end{pmatrix}$$

and the eigenvalues $\lambda_l, l = 1, 2$ satisfy $\lambda_1 \lambda_2 = 4a$. Finally, for any bivariate distribution with positive continuous density f at $\mathbf{1}$ and using the normalization $u_n(x) = \sqrt{4a}/(\pi f(\mathbf{1})) \cdot (x/n)$, we have by Theorem 8.3 for any $x > 0$ as $n \to \infty$

$$\Pr\{Y_1 = \min_{j \le n} g(\boldsymbol{X}_j) \le \frac{\sqrt{4a}}{\pi f(\mathbf{1})} \cdot \frac{x}{n}\} \to 1 - \exp(-x).$$

Hence we find $\alpha = -1$ for the Weibull limit distribution, which confirms the findings of Sect. 8.3.

8.5 Experiments Using Evolution Strategies

In this section we investigate the behavior of two other optimizers, where the application of the EVT is not as appropriate as for the random search optimizer discussed in Sect. 8.3. We consider two evolutionary strategies for $g : \mathbb{R}^2 \to \mathbb{R}$.

The first is a simple evolutionary algorithm described by the following scheme:

1. Choose a starting point $(a, b) \leftarrow (x_0, y_0)$ and a positive constant δ.
2. Generate six bivariate normally distributed random points $(a^i, b^i),\ i = 1, \ldots, 6$, with mean (a, b) and covariance matrix $\delta^2 \mathbf{I}$.
3. Let $(a, b) \leftarrow (x_1, y_1)$ such that $g(x_1, y_1) = \min_{i \le 6} g(a^i, b^i)$.
4. Repeat (2) and (3) a fixed number n of times (steps) and obtain a set of points $(x_j, y_j),\ j = 1, \ldots, n$.
5. Return the solution (x_{opt}, y_{opt}) such that $g(x_{opt}, y_{opt}) = \min_{j \le n} g(x_j, y_j)$.

The number of "offsprings" (a^i, b^i) in step 2 can also be larger than six, but in certain situations six is a good choice (see Bäck et al. (1991)). Often more offspring points, such as ten, are used, resulting in more stable self-adaption, but also in larger computing effort.

The second evolution strategy used is denoted by $(1, 6) - ES$ by Bäck et al. (1991) and differs from the first by the fact that the variance in step 2 is not constant but rather random and also adaptive in order to approach the optimum faster. In this case each of the random points in step 2 has its own covariance matrix $\delta_{j-1}^2 e^Z \mathbf{I}$,

Table 8.3: Evolution strategy 1 with $\delta = 0.1$ for Rosenbrock function, with the p-values of the A^2 test

Case	start point	steps	$\hat{\alpha}$	$\hat{\mu}$	$\hat{\sigma}$	p-value
1	$(-1, -1)$	100	−0.652	4.23e−5	0.011	0.00
2	optimum (1,1)	50	−0.783	3.57e−6	0.0039	0.03

where δ_{j-1}^2 is the variance used to generate the previous centre (x_{j-1}, y_{j-1}) and $Z \sim \mathrm{N}(0, \delta^2)$.

Obviously, the internal steps and random variables are dependent, thus we do not expect that $Y_i = g(x_{opt}, y_{opt})$ follows an extreme value distribution for small or moderate n. However, we may expect that the smallest of the generated Y_i can be modeled by a generalized Pareto distribution using the peaks over threshold (POT) approach of the extreme value theory.

The application of the goodness-of-fit test A^2 is not appropriate here for the POT approach, as mentioned. Although we tabulate the p-values in Tables 8.3 and 8.4 to show the inappropriateness of this test. We find values of the test A^2 with very small and large p-values in Table 8.3 and 8.4, but analyzing the QQ-plots in Figs. 8.5 and 8.6 we notice the same good fit for the smallest values and larger deviations for moderate or large values.

All the cases reported in Tables 8.3 and 8.4 were derived from samples of outcomes of size $k = 500$ and different number n of steps for the Rosenbrock function. It is necessary that the number of internal steps n is not small to obtain good results.

In most cases of Tables 8.3 and 8.4 the quality of the solutions is quite good as can be seen from the several estimated parameters. Note that the absolute values of

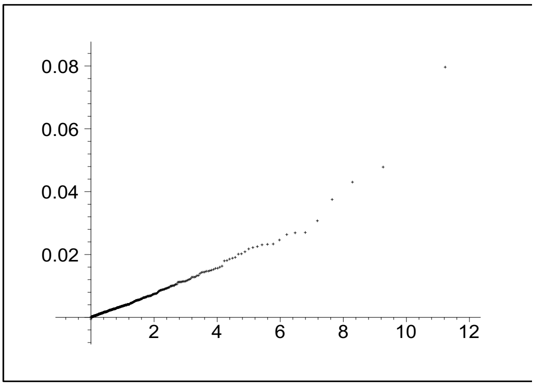


Fig. 8.5: QQ-plot of data of case 2 in Table 8.3, A^2-test: p-value = 0.03, Weibull distribution for minima with $\hat{\alpha} = -0.783$ (x-coordinate: Weibull quantiles, y-coordinate: sample quantiles)

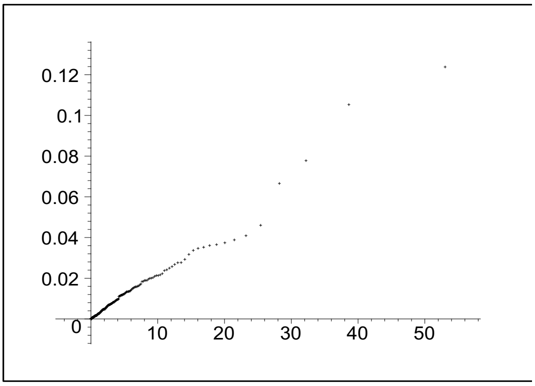


Fig. 8.6: QQ-plot of random search data of case 4 in Table 8.4, A^2-test: p-value = 0.50, Weibull distribution for minima with $\hat{\alpha} = -0.477$ (x-coordinate: Weibull quantiles, y-coordinate: sample quantiles)

Table 8.4: Evolution strategy 2 with starting standard deviation δ for Rosenbrock function, with the p-values of the A^2 test

Case	Start point	Steps	δ	$\hat{\alpha}$	$\hat{\mu}$	$\hat{\sigma}$	p-value
1	optimum (1,1)	10	0.01	−0.507	1.1e−6	0.0608	0.19
2	optimum (1,1)	10	1.00	−0.200	4.5e−12	0.0011	0.00
3	$(-1,-1)$	10	0.01	−0.530	2.1e−6	0.0985	0.08
4	$(-1,-1)$	50	0.01	−0.477	5.0e−8	0.0022	0.50

$\hat{\alpha}$ are smaller than those obtained in Sect. 8.3. However the fit with an extreme value distribution is neither satisfactory nor adequate when using the whole sample.

It is clear that the second evolution strategy performs better than the first one. We note that the second algorithm is able to reach the optimum in a smaller number n of steps (when starting away from the optimum) and that the shape parameter indicates also a greater clustering effect near the optimum. The different shape parameters of the evolutionary optimizers from the random search optimizer indicate that the dependence and the sequential behavior of the evolutionary strategies do not allow the classical extreme value distributions to be applied to the results.

Figures 8.5 and 8.6 show two samples which indicate our conjecture that the smallest values of the minima Y_i follow the extreme value behavior well. The simulated points are almost on a straight line near the minimum. This means that the smallest values Y_i can be well approximated by the lower tail of a Weibull distribution related to the generalized Pareto distribution with a negative shape parameter $\alpha = 1/\gamma$.

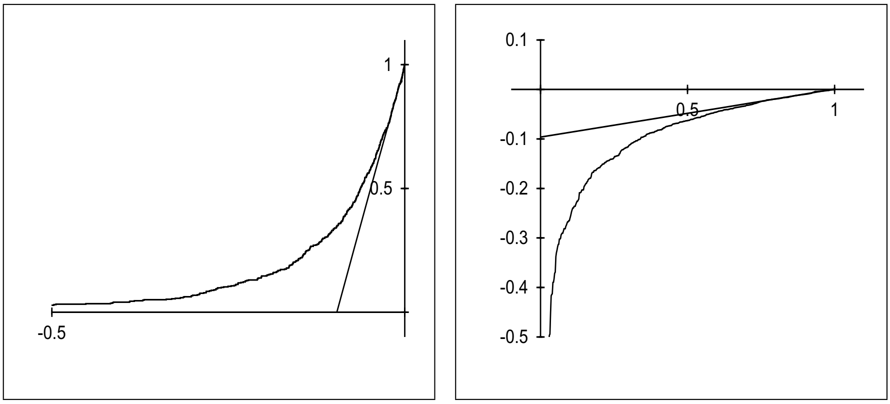


Fig. 8.7: Empirical cdf (left) and quantile function (right) of the 100 (largest) transformed values $-Y_i$ of the (first) evolutionary strategy for the Rosenbrock function, fitted to the cdf and quantile function of the generalized Pareto distribution (almost straight lines)

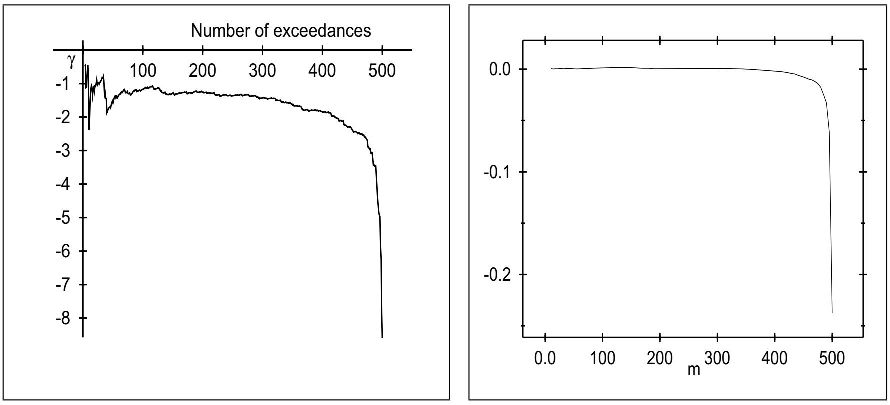


Fig. 8.8: Estimates of the shape parameter γ (left) and of the lower endpoint μ (right) (of the values Y_i) in relation to the number m of exceedances of $-Y_i$ for the Rosenbrock function

Figures 8.7 and 8.8 show the performance and the fit to a generalized Pareto distribution, the estimation of γ, and of the endpoint μ in relation to the number m of exceedances of $-Y_i$, based on $n = 100$ internal steps. Note that, since we use the software Xtremes for this modeling, we have to transform the minima to maxima values by a sign change. Hence the lower tail of the minima distribution is transformed to the upper tail of the maxima distribution.

The fit is excellent in the upper tail with $\hat{\gamma} = -1.1745$ for $m = 100$ for the outcomes $-Y_i$. The fit of the upper tail is also excellent if we select a larger m, say 200. Usually we should select $m/k \to 0$ as $k \to \infty$, by the POT theory. Typically, we choose m/k not larger than $5-10\%$ in the POT applications. However, in our application, where the observed values are already minima or maxima of some optimizer algorithm, we expect many more values in the neighborhood of the endpoint than we usually expect in a random sample of the whole distribution. So it is appropriate to apply a larger m than in the usual POT approach. This is supported by the estimates of γ, which are rather stable in a large range of m (for $50 \leq m \leq 250$), and also by the stable behavior of the endpoint estimation (for $50 \leq m \leq 350$). The estimate of the lower endpoint for the minimum is $\hat{\mu} = 0.0013$ with $m = 100$ and $\hat{\mu} = 0.00092$ with $m = 200$. The endpoint could be estimated also by a confidence interval.

If we select a smaller sample size k, we observe possibly smaller estimates of α. In the same way, if the number of internal steps n is selected smaller, we expect fewer values near the lower endpoint, which implies often a different estimate $\hat{\alpha}$. However, the estimation of the minimum μ improves by choosing a larger number of internal steps n, as well as the fit of the smallest observations to a generalized Pareto distribution.

We applied also the evolution algorithm to the second function and observed the same excellent behavior of the estimate $\hat{\gamma}$ and of the endpoint $\hat{\mu}$, as well as the fit of the generalized Pareto distribution, comparing the quantile and cumulative distribution functions. The same statements on the behavior of the evolution algorithm in the case of the Rosenbrock function can be repeated for this function.

Finally we applied the evolution algorithm and the POT approach to the third function *sinc*. Using 200 internal steps the global minima was found extremely accurately. Of the 500 simulated outcomes Y_i, only 112 values were larger than the global minima -0.217234 plus 0.00001. The largest deviation was fewer than 0.0002. So we applied the algorithm with less internal steps, $N = 20$. Figures 8.9 and 8.10 show again the excellent fit and the estimation of γ, which is stable in a large range of the number of exceedances m. The same holds also for the estimates of the upper endpoint, i.e., of the global minimum, which is almost not affected by the number of exceedances m if this is chosen between 50 and 250.

In general, evolution strategies are expected to perform better than random search. The obtained results clearly confirm this and also show that the improvement is reflected in the shape parameter of the distribution of outcomes. Note that some of the estimated α values are closer to zero than in the cases reported previously. As proved in Sect. 8.4, random search imposes a natural bound for the shape parameter which cannot be overcome by choosing different scales or starting points. Evolution strategies are able to overcome this bound and produce distributions which cluster much stronger near the optimum, thus resulting in better solutions.

We finish this section with an application of recently proposed tests which analyze the general assumption of EVT or POT. The tests are not designed to test the goodness-of-fit of a parametric class of distributions, as the A^2 test for the Weibull distributions. These tests are investigating the more general hypothesis that $F \in MDA(G_\gamma)$ for some γ, which is also the assumption for the POT approach. Here $F \in MDA(G_\gamma)$ means that F belongs to the $\underline{\text{m}}$ax-$\underline{\text{d}}$omain of $\underline{\text{a}}$ttraction of G_γ, a common abbreviation of EVT. These tests indicate whether the EVT or POT can be applied.

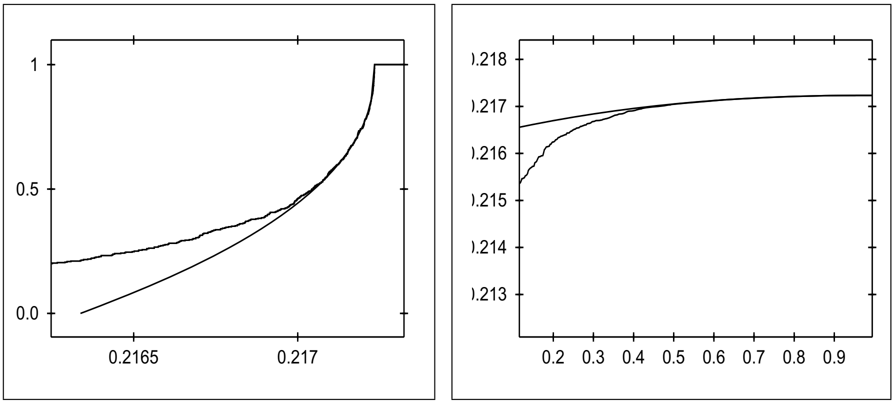


Fig. 8.9: Fit of the generalized Pareto distribution (smooth line) to the largest transformed values $-Y_i$ of the (first) evolutionary strategy for the sinc function with $n = 20$ internal steps, with $m = 200$

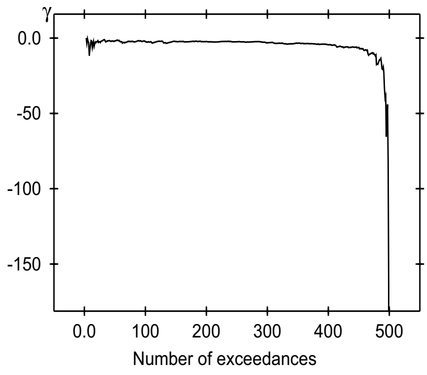


Fig. 8.10: Estimates of the shape parameter γ (of the values $-Y_i$) in relation to the number m of exceedances of $-Y_i$ for the sinc function

Two tests exist for this hypothesis. The test $E_k = E_{k,m}$ by Dietrich et al. (2002) is based on the comparison of the empirical quantile function with the estimated theoretical one $G^{-1}_{\hat{\gamma},\hat{\mu},\hat{\sigma}}$ using certain estimates of the parameters: let $X_i = -Y_i$ and $X_{i,k}$ denote the ordered values $X_i, i \leq k$; then

$$E_k = m \int_0^1 \left(\frac{\log X_{k-[mt],k} - \log X_{k-m,k}}{\hat{\gamma}_+} - \frac{t^{-\hat{\gamma}_-} - 1}{\hat{\gamma}_-}(1 - \hat{\gamma}_-) \right)^2 t^{\eta}\, dt \quad (8.2)$$

with some $\eta > 0$ and $\hat{\gamma}_+ = m_{1,m}$, an estimate for $\gamma_+ = \max\{\gamma, 0\}$, $\hat{\gamma}_- = 1 - 1/(2(1 - m_{1,m}^2/m_{2,m}))$, an estimate for $\gamma_- = \min\{\gamma, 0\}$. The distribution of E_k converges (as $k \to \infty$) to a distribution which can be determined only by simulations. The necessary quantiles were derived (Hüsler and Li (2006) for the corrected tables and more hints for the application and the selection of η). The asymptotic quantiles depend on η and γ. The parameters have to be estimated. This test applies the moment estimators $\hat{\gamma}_+, \hat{\gamma}_-$ of Dekkers et al. (1989). The result was stated for $\eta = 2$ by Dietrich et al. (2002), but it was extended for any $\eta > 0$ by Hüsler and Li (2006).

The test $T_k = T_{k,m}$ by Drees et al. (2006) is based on the comparison of the empirical cdf with the theoretical one $G_{\hat{\gamma},\hat{\mu},\hat{\sigma}}$, where $\hat{\gamma}$, $\hat{\mu}$, and $\hat{\sigma}$ are appropriate estimates:

$$T_k = m \int_0^1 \left(\frac{k}{m} \bar{F}_k \left(\hat{a}_{k/m} \frac{x^{-\hat{\gamma}} - 1}{\hat{\gamma}} + \hat{b}_{k/m} \right) - x \right)^2 x^{\eta - 2}\, dx, \tag{8.3}$$

where $\eta > 0$. The test statistic T_k converges (as $k \to \infty$) to a distribution which can only be derived by simulation and depends on η and γ. Here any $\sqrt{k}$-consistent estimator $(\hat{\gamma},\, \hat{a},\, \hat{b})$ of $(\gamma,\, a,\, b)$ could be used. The maximum-likelihood estimates are the typical appropriate candidates, but only for $\gamma > -1/2$.

An extension and the comparison of the two tests are treated by Hüsler and Li (2006) and Hüsler and Li (2007). They propose to take $\eta = 2$ for the test E_k and $\eta = 1$ (or also $\eta = 2$) for the test T_k. If γ seems to be positive, then both tests can be applied equally well to test H_0; otherwise the test E_k is preferable. Hence, the test E_k should be applied in an application for the optimizer.

Note that a significance would mean that "$F \notin MDA(G)$". Hence, the tests should give no indication against the null hypothesis, to apply the POT approach for the optimizer. The tests depend on the chosen number m of extreme order statistics. If m is chosen too large, then the tests reject the null hypothesis because intermediate order statistics are used for the assessment of the extreme tail. Thus the tests indicate also an appropriate choice of m for the application and estimation, as is shown in the following applications.

We apply the test E_k for the two cases discussed above, for the outcome values Y_i of the evolution optimizer for the Rosenbrock function and for the sinc function. The test statistics do not give a hint against the null hypothesis for m smaller than 150 for Rosenbrock function and 105 for the sinc function. The test seems to be more sensitive than our simple goodness-of-fit assessment based on the quantile or the distribution function of Figs. 8.7 and 8.9. So our estimation of the lower endpoint should be based on such an appropriate m.

8.6 Summary

We considered the behavior of optimizers, the random search optimizer and optimizers based on evolutionary strategies, for finding the minima or optimum of some continuous objective functions. We showed that the behavior of the outcomes of the optimizer can be modeled with the extreme value theory. The statistical estimation

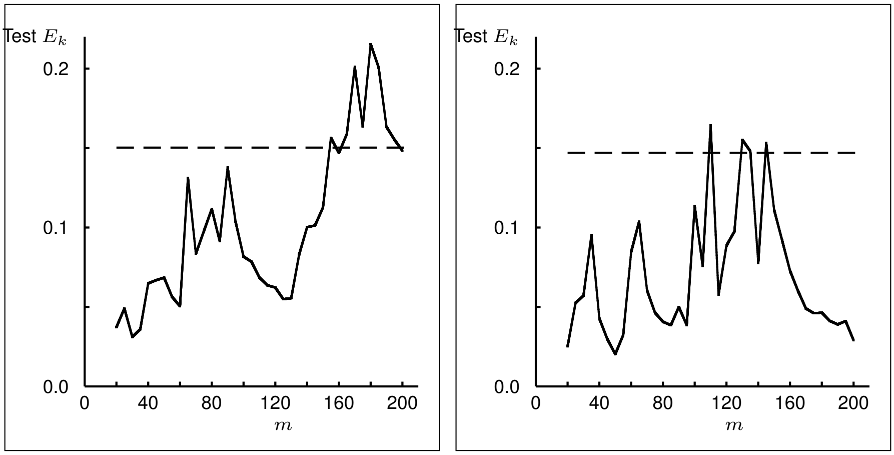


Fig. 8.11: Test E_k for the assessment of the POT approach for the two cases of the evolution optimizer, for the Rosenbrock function (left) and the sinc function (right). The dashed line indicates the 95% quantile of the test statistics

procedures of the extreme value theory allow us the estimation of the finite lower endpoint, the minimum of the objective function. This is demonstrated by some analytical results as well as some simulations for three chosen objective functions. The outcomes of the random search optimizer can be simply fitted to an generalized extreme value distribution, typically a Weibull distribution, if the internal number of steps is large. The evolutionary optimizer shows a different behavior because of the dependence of each internal step on the preceding one. We showed by simulations that this behavior can be simply described and modeled by a generalized Pareto distribution using the peaks over threshold (POT) approach. The Beta distribution is the particular distribution of the POT method to estimate the lower endpoint appropriately based on the outcomes of the evolutionary optimizer.

References

Bäck T, Hoffmeister F, Schwefel HP (1991) A survey of evolution strategies. In: Belew RK, Booker LB (eds) Genetic Algorithms: Proceedings of the Fourth International Conference at San Diego, Morgan Kaufmann, San Mateo, pp 2–9

Beirlant J, Goegebeur Y, Segers J, Teugels J (2004) Statistics of Extremes: Theory and Applications. Wiley, New York, NY

Dekkers A, Einmahl J, de Haan L (1989) A moment estimator for the index of an extreme-value distribution. Annals of Statistics 17:1833–1855

Dietrich D, de Haan L, Hüsler J (2002) Testing extreme value conditions. Extremes 5:71–86

Drees H, de Haan L, Li D (2006) Approximations to the tail empirical distribution function with application to testing extreme value conditions. Journal of Statistical Planning and Inference 136:3498–3538

Embrechts P, Klüppelberg C, Mikosch T (1997) Modelling Extremal Events for Insurance and Finance. Springer

Falk M, Hüsler J, Reiss R (2004) Laws of Small Numbers: Extremes and Rare Events, 2nd edn. Birkhäuser, Basel and Boston

Fisher R, Tippett L (1928) Limiting forms of the frequency distribution in the largest particle size and smallest number of a sample. In: Proceedings of the Cambridge Philosophical Society, pp 180–190

Fonseca C, Fleming P (1996) On the performance and comparison of stochastic multiobjective optimizers. In: Voigt HM, Ebeling W, Rechenberg I, Schwefel HP (eds) Parallel Problem Solving from Nature - PPSN IV, Springer, Lecture Notes in Computer Science, vol 1141, pp 584–593

de Haan L, Ferreira A (2006) Extreme Value Theory: An Introduction. Springer

Hüsler J, Li D (2006) On testing extreme value conditions. Extremes 9:69–86

Hüsler J, Li D (2007) Testing extreme value conditions with applications. In: Reiss RD, Thomas M (eds) Statistical analysis of extreme values, 3rd edn, Birkhäuser, Basel, pp 144–151

Hüsler J, Cruz P, Hall A, Fonseca C (2003) On optimization and extreme value theory. Methodology and Computing in Applied Probability 5:183–195

Leadbetter M, Lindgren G, Rootzen H (1983) Extremes and Related Properties of Random Sequences and Processes. Springer

Lockhart R, Stephens M (2006) Estimation and tests of fit for the three-parameter Weibull distribution. Journal of the Royal Statistics Society B(56):491–500

Reiss RD, Thomas M (1997) Statistical Analysis of Extreme Values with Application to Insurance, Finance, Hydrology and Other Fields. Birkhäuser, Basel

Resnick S (2007) Heavy-tail Phenomena. Springer

Smith R (1986) Extreme value theory based on the r largest annual events. Journal of Hydrology (86):27–43

Chapter 9
Exploratory Analysis of Stochastic Local Search Algorithms in Biobjective Optimization

Manuel López-Ibáñez, Luís Paquete, and Thomas Stützle

Abstract This chapter introduces two Perl programs that implement graphical tools for exploring the performance of stochastic local search algorithms for biobjective optimization problems. These tools are based on the concept of the empirical attainment function (EAF), which describes the probabilistic distribution of the outcomes obtained by a stochastic algorithm in the objective space. In particular, we consider the visualization of attainment surfaces and differences between the first-order EAFs of the outcomes of two algorithms. This visualization allows us to identify certain algorithmic behaviors in a graphical way. We explain the use of these visualization tools and illustrate them with examples arising from practice.

9.1 Introduction

Experiments in computer science often produce large amounts of data, mainly because experiments can be set up, performed, and repeated with relative facility. Given the amount of data, exploratory data analysis techniques are one of the most important tools that computer scientists may use to support their findings. In particular, specialized graphical techniques for representing data are often used to perceive trends and patterns in the data. For instance, there exist techniques for the extraction of relevant variables, the discovery of hidden structures, and the detection of outliers and other anomalies. Such exploratory techniques are mainly used during the design of an algorithm and when comparing the performance of various algorithms.

Luís Paquete
CISUC, Department of Informatics Engineering, University of Coimbra, Portugal
e-mail: paquete@dei.uc.pt

Manuel López-Ibáñez, Thomas Stützle
IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium
e-mail: manuel.lopez-ibanez@ulb.ac.be, stuetzle@ulb.ac.be

Even before testing more formal hypotheses, the algorithm designer has to find patterns in experimental data that provide further insights into new ways of improving performance.

In this chapter, we focus on the graphical interpretation of the quality of the outcomes returned by stochastic local search (SLS) algorithms (Hoos and Stützle 2005) for biobjective combinatorial optimization problems in terms of Pareto optimality. This notion of optimality is tied to the notion of dominance. We say that a solution dominates another one if it is at least as good as the latter for every objective and strictly better for at least one objective. For these problems, the goal is to find the set of *nondominated* solutions among all feasible solutions. The mapping of these solutions in the objective space is called the *Pareto-optimal front*. For the particular case of multiobjective combinatorial optimization problems (MCOPs), fundamental results about their properties and complexity are given by Ehrgott (2000).

Usually, each run of an SLS algorithm produces a nondominated set, a random set of mutually nondominated objective vectors that approximates the Pareto-optimal front of an MCOP. Currently, there are two widely used techniques for assessing the performance of these algorithms with respect to the solution quality: graphical examination of multiple outcomes and scalar quality indicators. Unfortunately, these two approaches present several drawbacks that have been discussed in the literature (Knowles and Corne 2002, Zitzler et al. 2003).

The empirical attainment function (EAF), formally described in Chap. 5, is seen in this chapter as a middle ground between directly plotting the complete output and the extreme simplification of quality indicators. The EAF is a summary of the outcomes of multiple runs of an SLS algorithm, and, at the same time, it is sufficiently complex to detect whether and where an algorithm is better than another. By plotting and comparing the EAFs of different SLS algorithms, we are able to pinpoint several performance behaviors that otherwise would be hidden when using other performance assessment approaches.

The chapter is organized as follows. Section 9.2 provides a basic introduction to SLS algorithms for multiobjective optimization in terms of Pareto optimality, and their performance assessment. Sections 9.3 and 9.4 introduce plotting techniques for exploring algorithm performance based on the EAF, and describe two Perl programs, `eafplot.pl` and `eafdiff.pl`. Section 9.5 presents three examples of applications of these programs. Finally, we present conclusions and further work in Sect. 9.6.

9.2 Stochastic Local Search for Multiobjective Problems

SLS algorithms iteratively search for good quality solutions using the local knowledge provided by the definition of a neighborhood or a set of partial solutions. Since they are based on a randomized search process, it is not expected that the same outcome is returned for different runs with different random seeds of the random number generator. *Metaheuristics* are general-purpose SLS methods that can be adapted

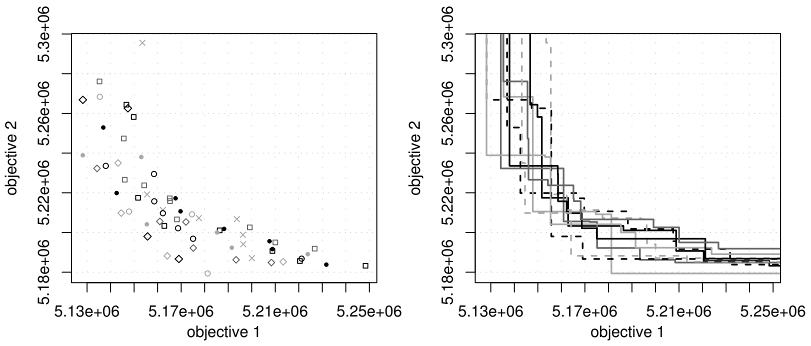


Fig. 9.1: Ten independent outcomes obtained by an SLS algorithm applied to an instance of a biobjective optimization problem. In the right plot, the same outcomes are shown but points belonging to the same run are joined with a line

to various optimization problems. Well known metaheuristics are simulated annealing, tabu search, iterated local search, variable neighborhood search, ant colony optimization, and evolutionary algorithms. An overview of these methods is given by Hoos and Stützle (2005).

Finding the Pareto-optimal front in MCOPs is known to be a hard challenge in optimization. Moreover, for many problems, the size of the Pareto-optimal front is too large to be enumerated (Ehrgott 2000). Therefore, depending on the time constraints, it could be preferable to have an approximation to the Pareto-optimal front in a reasonable amount of time. Such an approximation is always a *nondominated set* , that is, a set of objective vectors that are mutually nondominated.

SLS algorithms have been shown to be state-of-the-art methods for generating very good approximations to the Pareto-optimal front for many MCOPs. As a result, when comparing two SLS algorithms, we need to compare nondominated sets that are in most cases incomparable in the Pareto sense. There are mainly two approaches for summarizing and comparing SLS algorithms with respect to solution quality: direct examination of multiple nondominated sets and scalar quality indicators.

As an example of direct examination of the outcomes of an SLS algorithm, we plot in Fig. 9.1 the outcomes obtained by ten runs of the same SLS algorithm applied to an instance of a biobjective optimization problem. On the left plot, we plot the objective vectors as points. Points with the same shade of gray and shape were obtained in the same run. In the right plot, objective vectors from the same run are joined with staircase lines delimiting the area dominated by them. Even with only ten runs, it is difficult to visualize the algorithm behavior. With a larger number of runs, directly plotting the outcomes quickly becomes impractical. A direct comparison of the outcomes of different algorithms is similarly difficult.

On the other extreme are scalar quality indicators, which are scalar values computed for each nondominated set (or pairs of nondominated sets) (Knowles and Corne 2002, Zitzler et al. 2003). Quality indicators are surrogate measures of particular quality aspects of the nondominated sets (e.g., closeness to the best-known solutions, spread, diversity) that are generally acknowledged as desirable. Hence, several quality indicators are often examined simultaneously, but this, in turn, complicates the interpretation of results. The values returned by quality indicators often do not reflect by how much and in which aspects a nondominated set is better than another. Moreover, recent theoretical work has shown that many quality indicators may provide an answer that contradicts the most basic and deterministic assertions of performance. More details are given by Knowles and Corne (2002) and Zitzler et al. (2003).

A different perspective on performance assessment of SLS algorithms for MCOPs is given by the attainment function approach (Grunert da Fonseca et al. 2001). Chapter 5 provides a theoretical overview on attainment functions. In this chapter, we focus on the first-order attainment function, which represents the probability of an algorithm finding at least a solution whose objective vector dominates or is equal to an arbitrary vector in the objective space in a single run. In practice, this probability is unknown, but it can be estimated empirically from the outcomes obtained in several independent runs of an SLS algorithm, in a way analogous to the estimation of multivariate distribution functions. This statistical estimator is called the (first-order) empirical attainment function (EAF) (Grunert da Fonseca et al. 2001).[1]

In the biobjective case, the EAF is both fast to compute and easy to visualize. We will consider two different visualizations. First, plots of the $k\%$-attainment surfaces are used to characterize the behavior of a single SLS algorithm. Second, the performance of two SLS algorithms is compared by plotting the location of the differences with respect to their EAFs.

9.3 Examination of the Attainment Surfaces

Fonseca and Fleming (1996) proposed the notion of *attainment surface*, which corresponds to a boundary which separates the objective space into two regions: those objective vectors that are attained by (that is, dominated by or equal to) the outcomes returned by the SLS algorithm, and those that are not. This notion is formalized in the concept of $k\%$-attainment surface, which is the line separating the objective space attained by $k\%$ of the runs of an SLS algorithm. In other words, the $k\%$-attainment surface corresponds to the $k/100$ percentile of the empirical frequency distribution. For example, the *median* attainment surface delimits the region attained by 50% of the runs. Similarly, the *worst* attainment surface delimits the region attained by all runs (100%-attainment surface), whereas the *best* attainment surface corresponds to the limit between the region attained by at least one run and the objective vectors never attained by any run.

Given m runs, the computation of the EAF is equivalent to the computation of all $k\%$-attainment surfaces with $k = i \cdot 100/m$, $i = 1, \ldots, m$. In fact, the $k\%$-

[1] We will always refer to the first-order EAF simply as EAF.

attainment surface is sufficiently defined by the nondominated objective vectors from the set of all objective vectors that are attained by $k\%$ of the runs.

The attainment surfaces allow to summarize the behavior of an SLS algorithm in terms of the location of the objective vectors obtained. For example, if we were interested in the objective vectors that are attained by at least half of the runs, then we could examine the median attainment surface. Similarly, the worst-case results of an algorithm are described by the worst attainment surface, whereas the best results ever achieved are given by the best attainment surface. Sections 9.3.2 and 9.5.1 give examples.

9.3.1 The `eafplot.pl` Program

The program `eafplot.pl` is a Perl program that produces a plot of attainment surfaces given an input file that contains a number of nondominated sets. Input files may contain multiple sets of nondominated objective vectors. Each objective vector is given in a line as two columns of floating-point numbers. Different sets are separated by at least one blank line. If several input files are given, then one plot is produced for each input file, all plots having the same range on the axes. The plots produced are encapsulated postscript (EPS) files.

These programs require a Perl installation, the statistical environment R (R Development Core Team 2008), and an external program for computing the EAF.[2]

The attainment surfaces plotted by `eafplot.pl` can be specified in several ways:

- By default, `eafplot.pl` plots the best, median, and worst attainment surfaces.
- Option `-iqr` plots the 25%, 50% (median), and 75% attainment surfaces.
- Option `-percentile=INT[,INT]` plots the given percentiles of the attainment surface. For example, `eafplot.pl -percentile=25,50,75` is equivalent to `eafplot.pl -iqr`.
- Option `-extra=FILE` will add objective vectors from `FILE` to the plot as points. This may be useful for comparing the outcome of an SLS algorithm against a reference set.

The program accepts other parameters that are not discussed in this chapter but are explained by the option `-help`.

[2] The program for computing the EAFs provided by us is based on the original code written by Carlos M. Fonseca available at `http://www.tik.ee.ethz.ch/pisa/`.

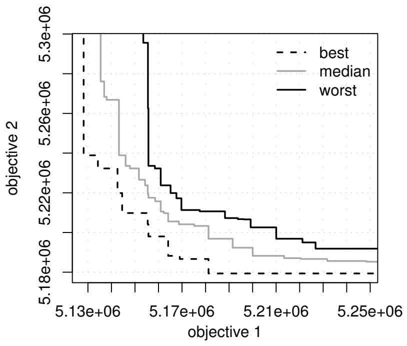


Fig. 9.2: Best, median, and worst attainment surfaces for the data described in Fig. 9.1

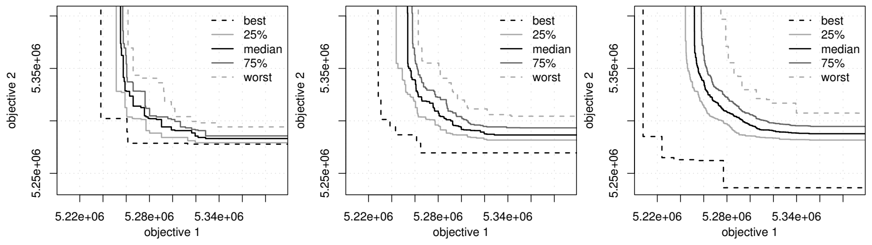


Fig. 9.3: Three plots of attainment surfaces for 15 (left), 50 (middle), and 200 (right) independent runs of the same algorithm on the same problem instance

9.3.2 Example Application of `eafplot.pl`

Given the input data shown in Fig. 9.1, the corresponding best, median, and worst attainment surfaces are shown in Fig. 9.2. This plot was generated by the command

```
   eafplot.pl example1_dat
```

As an alternative to the best and worst attainment surfaces, one may prefer to plot other percentiles that are more robust with respect to the number of runs. The dependence of the best and worst attainment surfaces on the number of runs is illustrated by Fig. 9.3, where the same algorithm is run 15 (left), 50 (middle), and 200 (right) times with different random seeds. As more runs are performed, the locations of the best and worst attainment surfaces change strongly, while the locations of the 25% and 75% attainment surfaces are rather stable. It is well known from classical statistics that the sample best and worst are biased estimators for the population best and worst. The three plots in Fig. 9.3 were produced by running:

```
eafplot.pl --best --median --worst  \
    --percentiles=25,75  r15_dat r50_dat r200_dat
```

9.4 Examining the Differences Between EAFs

The EAF is also the basis for a graphical technique that gives visual information on the pairwise comparison of two SLS algorithms. The main idea is to plot the location of the differences between the outcomes of two algorithms with respect to their corresponding EAFs. The EAF of an algorithm estimates the probability of attaining each point in the objective space. If the difference of the estimated probability values of two SLS algorithms at a certain point is large, this indicates better performance of one algorithm over another at that point. The sign of the difference gives information about which algorithm performed better.

The differences between the EAFs of two algorithms can be computed by first computing the EAF of the union of the outcomes of both algorithms. Then, for each point in the objective space where the value of the EAF changes, one needs to compute the value of the EAF of the first algorithm at that point minus the value of the EAF of the second algorithm. This can be done by counting how many runs of each algorithm attained that point. Finally, positive and negative differences are plotted separately, and the magnitudes of the differences between the EAFs are encoded using different shades of grey: the darker a point, the larger the difference.

Figure 9.4 illustrates this performance assessment method. The two plots in the top part of Fig. 9.4 give the EAFs associated with two algorithms that were run several times with different random seeds on the same problem instance. Points in the EAFs are assigned a gray level according to their probability. In addition, we plot four different attainment surfaces. The lower line on both plots connects the best set of points attained over all runs of both algorithms (*grand* best attainment surface), and the upper one the set of points attained by any of the runs (*grand* worst attainment surface). Any differences between the algorithms are contained within these two lines. The dashed line corresponds to the median attainment surface of each algorithm, which is given to facilitate the comparison of the two sides of the plot.

The bottom plots of Fig. 9.4 show the location of the differences between the EAFs of the two algorithms.[3] On the left are shown points where the EAF of algorithm 1 is larger by at least 20% than that of algorithm 2, and on the right are given the differences in the opposite direction (positive differences between the EAF of algorithm 2 over the one of algorithm 1). The amount of the differences is encoded in a grey scale shown in the legend of the plot. To facilitate comparison, the same attainment surfaces are plotted as for the top plots. From these plots, we can observe that algorithm 1 performs better in the center and towards the minimization of objective 1, whereas algorithm 2 performs better towards high-quality solutions for the second objective (low values on the y-axis). Note that these differences in performance would be ignored by most scalar quality indicators.

The program `eafdiff.pl` is a Perl program that takes two input files, each of which contains a number of nondominated sets, and produces a plot of the differ-

[3] The same information could be provided within one plot by using different colors for positive and negative differences.

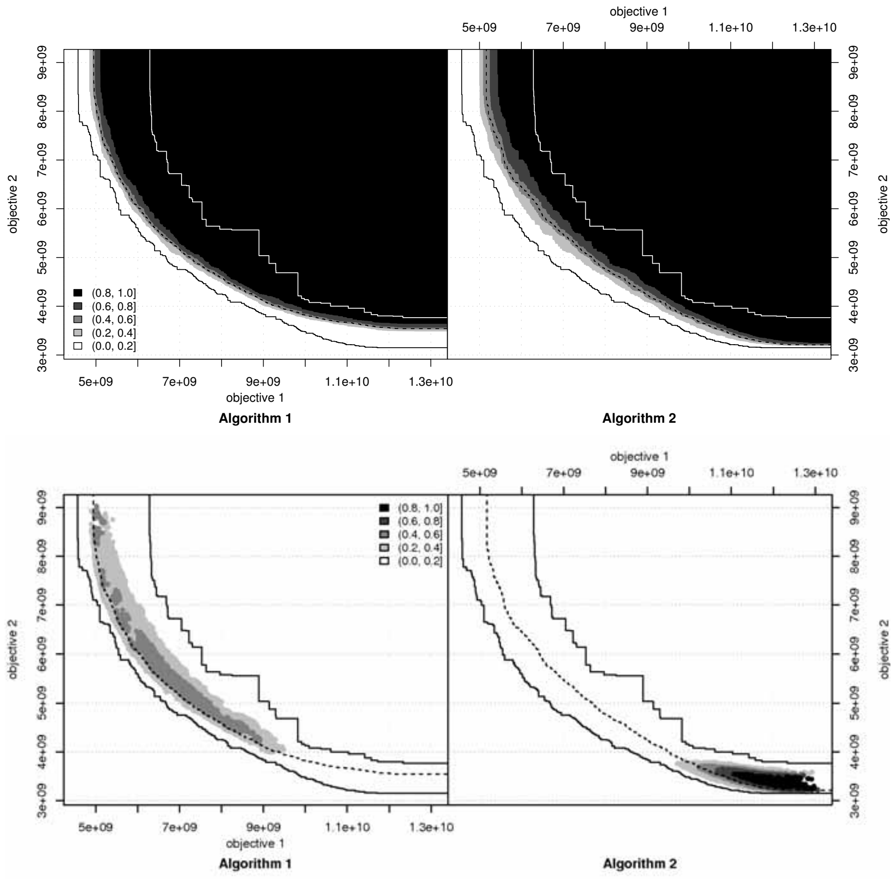


Fig. 9.4: Visualization of the EAFs associated with the outcomes of two algorithms (*top*) and the corresponding differences between the EAFs (*bottom left*: differences in favor of algorithm 1; *bottom right*: differences in favor of algorithm 2). In the top, the gray level encodes the value of the EAF. In the bottom, the gray level encodes the magnitude of the observed difference

ences between the first-order EAFs of the two input files. It can produce two types of plots:

- A side-by-side plot of the full EAF of each of the input files. This type of plot can be requested by using the option `-full`. For example, the top plot of Fig. 9.4 was produced by the commandline:

```
eafdiff.pl --full --left="Algorithm 1" ALG_1_dat \
                  --right="Algorithm 2" ALG_2_dat
```

- A side-by-side plot of the differences in the EAFs between the two input files. This is the default. For example, the bottom plot of Fig. 9.4 was generated by:

```
eafdiff.pl --left="Algorithm 1" ALG_1_dat \
```

Table 9.1: Output files produced by `eafdiff.pl` given input files `file1` and `file2`

Output plot	*file1-file2*.eps
Grand best attainment surface	*file1-file2*.best
Grand worst attainment surface	*file1-file2*.worst
Differences between EAFs	*file1-file2*.diff
Full EAF of input file `fileX`	*fileX*.eaf
Median attainment surface of file `fileX`	*fileX*.med

```
        --right="Algorithm 2" ALG_2_dat
```

By default, `eafdiff.pl` plots also the grand best and grand worst attainment surfaces as solid black lines, and the median attainment surface corresponding to each input file as dashed lines. The program accepts other parameters that are not discussed in this chapter but are explained by the option `-help`.

Apart from the plot *file1-file2*.eps, the `eafdiff.pl` program produces several output files. These are listed in Table 9.1.

As for the computation time required by `eafdiff.pl`, in the example shown in Fig. 9.4, each of the two data sets contains 90 runs with an average of 500 objective vectors per run. The generation of the plot from these data sets required less than 10 seconds of computation time on a Intel Core™ 2 CPU with 1.83 GHz. The computation of the EAF in two dimensions is linear with respect to both the number of runs and the total number of points. Plotting is also linear; however, in practice, the computation time required by the plotting functions of R and writing out the output files are the slowest parts of the procedure.

9.5 Examples

In this section, we illustrate the use of the graphical techniques on several examples. As we will see, these tools allow us to discover particular algorithm behaviors.

9.5.1 Effect of Problem Structure

Problem structure has a clear effect on algorithm performance. In multiobjective optimization, the correlation between the objectives is often an example of this, since we expect that a large positive correlation between the objectives may induce a small Pareto-optimal front, and vice versa.

We reproduce here experiments described by López-Ibáñez et al. (2006). We consider two instances of the biobjective quadratic assignment problem (BQAP)

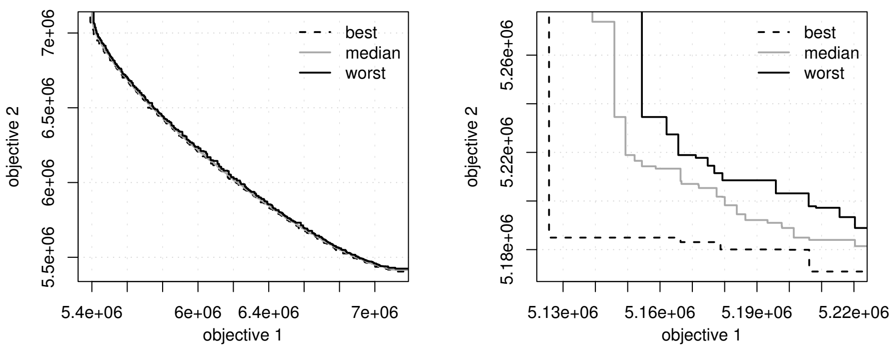


Fig. 9.5: The same algorithm is applied to two BQAP instances with correlation -0.75 (left) and 0.75 (right). The plots show the best, median, and worst attainment surfaces of ten runs

with different correlations between the flow matrices, which translate into different correlations between the corresponding objectives; see López-Ibáñez et al. (2006) for a more thorough explanation of this problem.

We plot in Fig. 9.5 the best, median, and worst attainment surfaces of the outcomes obtained by the same algorithm when applied to two BQAP instances with correlation -0.75 (left) and 0.75 (right). In each case, ten independent runs of the algorithm were performed with different random seeds. Even thought both instances are similar in terms of size and range of values, the range of the nondominated sets obtained for correlation -0.75 (left) is much wider than for correlation 0.75 (right), as can be seen in the range of the objective values in each of the plots. The plots show a strong effect of the correlation on the location of the outcomes obtained by the algorithm.

The two plots in Fig. 9.5 were produced by running:

```
eafplot.pl n75_dat

eafplot.pl p75_dat
```

9.5.2 Differences in Algorithmic Performance

Several approaches to biobjective problems consist of solving several weighted scalarizations of the objective function vector. Usually, the components of the weight vectors are real numbers in $[0, 1]$ and sum to one. With a subset of weight vectors evenly distributed in the full set of possible weight vectors, we may expect to find a well-spread set of objective vectors.

Two distinct algorithmic behaviors may be expected either by increasing the number of weights or by running the underlying stochastic algorithm for a longer time for each scalarization. Intuitively, by increasing the number of weights, the algorithm should be able to obtain a larger number of nondominated objective vectors distributed along the Pareto front, which gives a better approximation of the shape of the Pareto-optimal front. On the other hand, by giving more time to each scalarization, the resulting nondominated objective vector is typically of a higher quality. Because of limits in computation time, the algorithm designer has to examine the trade-off between these two settings in order to improve solution quality.

We describe an experiment that examines the trade-off between different parameter settings of weighted robust tabu search (WRoTS) for the biobjective QAP, such as described by López-Ibáñez et al. (2006). In WRoTS, several scalarizations of the BQAP objective function vector are solved by repeated runs of the (single-objective) robust tabu search (RoTS) algorithm (Taillard 1991). A single run of the RoTS algorithm is stopped after $l \cdot n$ iterations, where n is the instance size and l is a parameter. Each scalarization uses a different weight, taken from a set of maximally dispersed weights (Steuer 1986). We denote the number of weights by w. Upon termination of the main search process, all solutions returned are filtered to obtain a set of nondominated objective vectors.

Figure 9.6 shows the differences in EAFs between two algorithm configurations of WRoTS. The left plot shows differences in favor of performing few scalarizations and long runs of RoTS ($l = 100$, $w = 10$), whereas the right plot shows differences in favor of performing many short runs of RoTS ($l = 10$, $w = 100$). Note that the total number of iterations performed by each algorithm is the same. There are very strong differences in three particular regions of the right plot (many scalarizations and short runs of RoTS). These regions are probably difficult to attain with the coarse set of weights examined by the left configuration. On the other hand, the use of a larger number of iterations of RoTS (left plot) does not lead to better individual objective vectors, except for the extreme objective values.

The plots in Fig. 9.6 were generated by the command:

```
eafdiff.pl --left="WRoTS, l=100, w=10"  \
           --right="WRoTS, l=10, w=100" \
           wrots_l100w10_dat wrots_l10w100_dat
```

9.5.3 Biased Behavior

An algorithm may be focusing too much on a particular region of the objective space in detriment to other equally relevant regions, which may be due to some algorithmic choice. In this example, we present a case in which an algorithm is more biased towards one objective.

We describe an algorithm that was proposed by Paquete and Stützle (2003), called two-phase local search (TPLS): the first phase consists of finding a good solution to one single objective, using an effective single objective algorithm. This phase provides the starting solution for the second phase, in which a local search algorithm

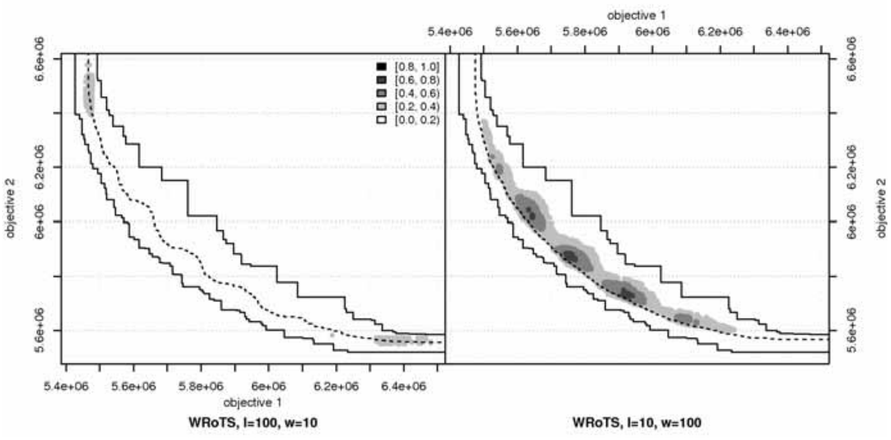


Fig. 9.6: EAF differences for two configurations of WRoTS. The left plot shows differences in favor of long runs of RoTS and few scalarizations ($l = 100$, $w = 10$); the right plot shows differences in favor of short runs of RoTS and many scalarizations ($l = 10$, $w = 100$)

is applied to a sequence of different scalarizations of the objectives. The underlying idea for the second phase is that successive scalarizations are treated as a chain: a scalarization slightly modifies the emphasis given to the different objectives when compared with the previous scalarization; the local search for each scalarization is started from the local optimum solution that was returned by the previous scalarization. The main question is whether this strategy can have comparable performance to a restart strategy (restart), which starts from a randomly generated solution at every scalarization.

The experimental analysis was performed for the biobjective traveling salesman problem . Both algorithms, TPLS and restart, have the same underlying stochastic local search, an iterated local search (Stützle and Hoos 2001).

Figure 9.7 shows the EAF differences between TPLS and restart. TPLS is able to obtain good solutions with respect to the second objective. However, TPLS is not able to improve the solutions obtained by restart with respect to the first objective.

The plots in Fig. 9.7 were obtained with the command:

```
eafdiff.pl --left="TPLS" --right="Restart" tpls rest
```

9.6 Summary and Outlook

We have described in this chapter graphical techniques for summarizing and comparing the quality of SLS algorithms for biobjective problems in terms of Pareto optimality. We have also described two programs that implement these techniques. In addition, examples of the usage of these programs were provided throughout the

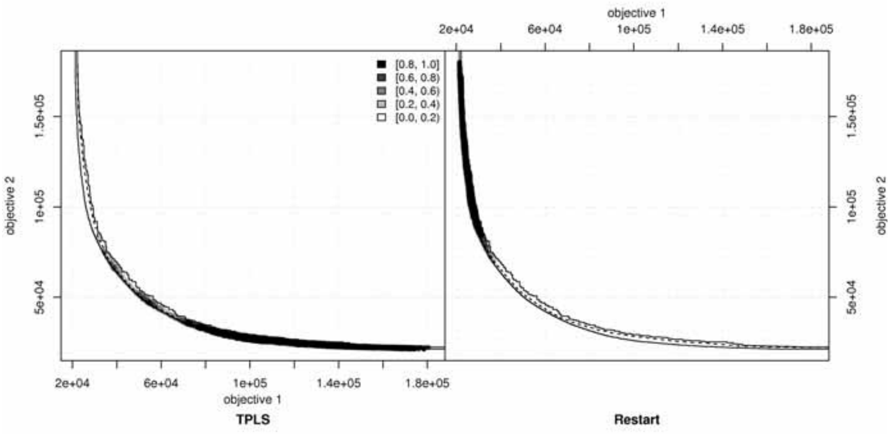


Fig. 9.7: EAF differences for TPLS versus restart

chapter. These examples can be reproduced by using the programs and data available at `http://iridia.ulb.ac.be/supp/IridiaSupp2009-002/`.

These graphical techniques are based on the first-order EAF for biobjective optimization problems. For more than two objectives, the graphical technique of parallel coordinates (Inselberg 1985) has been used by Paquete and Stützle (2009). However, the interpretation of the plots is more difficult than in the biobjective case. Therefore, other ways to present the information given by the first-order EAF may be worth investigating. Finally, new techniques could be developed based on the information provided by higher-order EAFs (Fonseca et al. 2005).

Acknowledgements This work was supported by the META-X project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium. Thomas Stützle acknowledges support from the Belgian F.R.S.-FNRS, of which he is a Research Associate. The authors also acknowledge Carlos M. Fonseca for providing us the code for computing the EAF as well as for helpful discussions on the main topic of this chapter.

References

Ehrgott M (2000) Multicriteria Optimization, Lecture Notes in Economics and Mathematical Systems, vol 491. Springer, Heidelberg, Germany

Fonseca CM, Fleming P (1996) On the performance assessment and comparison of stochastic multiobjective optimizers. In: Voigt HM, Ebeling W, Rechenberg I, Schwefel HP (eds) Proceedings of PPSN-IV, Fourth International Conference on Parallel Problem Solving from Nature, Springer, Lecture Notes in Computer Science, vol 1141, pp 584–593

Fonseca CM, Grunert da Fonseca V, Paquete L (2005) Exploring the performance of stochastic multiobjective optimisers with the second-order attainment function. In: Coello CC, Aguirre AH, Zitzler E (eds) Evolutionary Multi-criterion Optimization (EMO 2005), Springer, Lecture Notes in Computer Science, vol 3410, pp 250–264

Grunert da Fonseca V, Fonseca CM, Hall A (2001) Inferential performance assessment of stochastic optimizers and the attainment function. In: Zitzler E, Deb K, Thiele L, Coello CC, Corne D (eds) Evolutionary Multi-criterion Optimization (EMO 2001), Springer, Lecture Notes in Computer Science, vol 1993, pp 213–225

Hoos H, Stützle T (2005) Stochastic Local Search – Foundations and Applications. Morgan Kaufmann Publishers, San Francisco, CA

Inselberg A (1985) The plane with parallel coordinates. Visual Computer 1(4):69–91

Knowles J, Corne D (2002) On metrics for comparing non-dominated sets. In: Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02), IEEE Press, Piscataway, NJ, pp 711–716

López-Ibáñez M, Paquete L, Stützle T (2006) Hybrid population-based algorithms for the bi-objective quadratic assignment problem. Journal of Mathematical Modelling and Algorithms 5(1):111–137

Paquete L, Stützle T (2003) A two-phase local search for the biobjective traveling salesman problem. In: Fonseca CM, Fleming P, Zitzler E, Deb K, Thiele L (eds) Proceedings of the Evolutionary Multi-criterion Optimization (EMO 2003), Springer, Lecture Notes in Computer Science, vol 2632, pp 479–493

Paquete L, Stützle T (2009) Design and analysis of stochastic local search algorithms for the multiobjective traveling salesman problem. Computers and Operations Research 36(9):2619–2631

R Development Core Team (2008) R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, URL `http://www.R-project.org`

Steuer RE (1986) Multiple Criteria Optimization: Theory, Computation and Application. Wiley Series in Probability and Mathematical Statistics, John Wiley & Sons, New York, NY

Stützle T, Hoos H (2001) Analysing the run-time behaviour of iterated local search for the travelling salesman problem. In: Hansen P, Ribeiro C (eds) Essays and Surveys on Metaheuristics, Kluwer Academic Publishers, Boston, MA, pp 589–611

Taillard ÉD (1991) Robust taboo search for the quadratic assignment problem. Parallel Computing 17:443–455

Zitzler E, Thiele L, Laumanns M, Fonseca CM, Grunert da Fonseca V (2003) Performance assessment of multiobjective optimizers: An analysis and review. IEEE Transactions on Evolutionary Computation 7(2):117–132

Part III
Algorithm Configuration and Tuning

Chapter 10
Mixed Models for the Analysis of Optimization Algorithms

Marco Chiarandini and Yuri Goegebeur

Abstract We review linear statistical models for the analysis of computational experiments on optimization algorithms. The models offer the mathematical framework to separate the effects of algorithmic components and instance features included in the analysis. We regard test instances as drawn from a population and we focus our interest not on those single instances but on the whole population. Hence, instances are treated as a *random factor*. Overall these experimental designs lead to *mixed effects linear models*. We present both the theory to justify these models and a computational example in which we analyze and comment on several possible experimental designs. The example is a component-wise analysis of local search algorithms for the 2-edge-connectivity augmentation problem. We use standard statistical software to perform the analysis and report the R commands. Data sets and the analysis in SAS are available in an online compendium.

10.1 Introduction

Linear statistical models are well-developed mathematical tools for the separation of effects in the observed results of an experiment. Among them, there is the classical analysis of variance (ANOVA). scientific disciplines and also in the field of optimization. In operations research, application examples to test mathematical programming software go back to the late 1970s, see, e.g., Zanakis (1977), Lin and Rardin (1979), Coffin and Saltzman (2000); while in computer science and in test-

Marco Chiarandini
Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark e-mail: marco@imada.sdu.dk

Yuri Goegebeur
Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark, and Research Group Quantitative Psychology and Individual Differences, K.U.Leuven, Belgium e-mail: yuri.goegebeur@stat.sdu.dk

ing heuristic and evolutionary computation methods their use can be traced back to the late 1990s. Prominent articles in this case are those by Barr et al. (1995), McGeoch (1996), Rardin and Uzsoy (2001), and Czarn et al. (2004). However, only a very small number of articles, relative to those published in these fields, report use of these statistical methods. This fact might be explained by two factors: the need for a background in statistics and experimental design techniques in order to correctly apply and fully understand the results provided; and the presence of underlying assumptions that make the researcher in computer science or operations research sceptical about the real applicability of these methods in the field of optimization. The aim of this chapter is to introduce the reader to the use of linear statistical models in the cases where they can be applied. We aim to present the basic theory behind the methods, their practical application by means of publically available software and the possible outcomes. We go perhaps to a deeper level of detail compared with previous publications in this field, hoping to facilitate future applications. However, we do not aim to remove completely the two barriers above: understanding of statistics and a careful investigation of applicability to each specific case are necessary preconditions. Knowledge of the material in the appendix of this book might be required to follow this chapter.

We emphasize that our intention is to present these tools as complementary to and not substitutes for the current practice of reporting numerical results on benchmark instances with appropriate tables. This practice is indeed helpful to guarantee comparability and verifiability of results. The methods in this chapter are however desirable for scientific experimental analysis, where the interest is in explaining the causes of success of a certain optimization approach rather than in mere comparative studies; see Hooker (1996) for a discussion on these guidelines.

To illustrate the application of the statistical tools we use a case example in which we study heuristic algorithms for a graph problem: finding the cheapest augmentation of arcs that make a network 2-edge-connected (Bang-Jensen et al. 2009). The heuristics are local search algorithms (Michiels et al. 2007) obtained by the combination of some specific components, which may be *qualitative*, like for the presence or not of an algorithmic step or *numerical*, like for parameters that assume real values. Our interest is in understanding the contribution of these components.

In statistical terms, these components are called *factors*. The interest is in the effects of the specific *levels* chosen for these factors. Hence, we say that the levels and consequently the factors are *fixed*. Moreover, when for two factors, every factor level of a factor appears with every factor level of another factor we say that the two factors are *crossed*. We restrict ourselves to analyze the effect of these factors on a univariate measure of performance, namely the quality of the solutions returned by the algorithm at termination. Multivariate analysis are however also possible by extensions of these methods; we refer to Johnson and Wichern (2007) for an overview of these.

Typically, the researcher takes a few instances for the problem at hand and collects the results of some runs of the algorithms on these instances. The instances are treated as *blocks* and all algorithms are run on each single instance. Results are therefore *grouped* per instance. The instances are chosen at random from a large set of possible instances of the problem, and the interest of the researcher is not just on the performance of the algorithms on those specific instances chosen, but rather on the generalization of the results to the entire population of instances. In statistical terms, instances are also levels of a factor. However, this factor is of a different nature from the fixed algorithmic factors described above. Indeed, the levels are chosen at random and the interest is not in these specific levels but in the population from which they are sampled. We say that the levels and the factor are *random*.

Further, it might be possible to stratify the instances according to some characteristics or features that are easily retrievable. The researcher might then be interested in studying the influence of these characteristics on the performance of the algorithms. Instance characteristics can be regarded as fixed factors, because we can control them and the interest is on the specific levels. However, in such a study another issue arises: the instances at different levels of the instance factors are different, that is, they are sampled from different populations. In other terms, the random factor does not cross like all other factors, but it is instead *nested* within some of them.

In statistics, the effects described are modeled as linear combinations, and mathematical theory has been developed to make inferences about the populations on the basis of the results observed in the samples. The mixed nature of the factors leads to so-called *nested linear mixed models*; see for instance Molenberghs and Verbeke (1997), Montgomery (2005), and Pinheiro and Bates (2000). These designs, which are typical of the context of optimization, are nontrivial designs and go beyond the classical multifactorial ANOVA, where all factors are instead treated as fixed. As we will see, the mathematical formula involved and the inference derived are different in the case of mixed-effects models and this may lead to a different inference. In our practical application we will give an example where this difference clearly arises. To the best of our knowledge, only Lin and Rardin (1979) make clear reference to the nesting issue while in all other articles that we reviewed the mixed nature of the factors is not emphasized or is ignored.

This whole chapter is based on the assumption that additive linear models and normal distributions are appropriate to describe the experimental data. This is clearly a strong assumption that is often not met in experiments involving optimization algorithms. In fact, the example that we develop in the second part of the chapter was selected out of three, where the other two did not pass a diagnostic analysis on the assumptions. The arguments in defense of these tools also when assumptions are not met are the proven robustness of F-ratio tests in the analysis of variance method (Montgomery 2005) and that small adjustments of the data, like increases in the number of observations, removal of outliers, and opportune data transformations (e.g., log transformation) may contribute to meeting the assumptions. Our point of view is that, even when assumptions are not met, these tools can be very useful exploratory devices to look into the data. Extensions and generalizations that remove the need for these assumptions exist but for reasons of space we will not review them here.

The approach that we take to statistical inference is the classical one from statistics in which experiments are fully designed a priori. Even though differences among the entities studied always exist, we assume as correct the conservative hypothesis of no differences, and distinguish between *statistical differences* and *practically meaningful differences*. In this sense, we define a minimal effect size that is relevant in practice and derive the amount of data necessary to achieve a statistical power of 0.80 at a given level of significance of 0.05. We acknowledge that there are other ways to address the issue of sample size determination in experimental design. We refer, for example, to Chap. 13 for a discussion and description of the *sequential testing* approach.

The chapter is organized as follows. In Sect. 10.2, we formalize the problem of inference and the experimental designs, and we provide analytical support for the use of mixed models. We then review the theoretical background of the analysis. A reader primarily interested in the practical application of these methods may skip this part or consider it only when referenced back in Sect. 10.4. In Sect. 10.3, we introduce the application example on the 2-edge-connectivity problem. In Sect. 10.4, we develop the example producing an extended numerical analysis that reflects the mathematical background and the organization of Sect. 10.2. With the aim of facilitating reproduction, we report explicitly, in this section, the commands for the analysis in R, the free software environment for statistical computing (R Development Core Team 2008). We conclude in Sect. 10.5 with a summary and pointers to further developments that could be helpful in similar studies.

10.2 Experimental Designs and Statistical Modeling

In the most basic design, the researcher wishes to assess the performance of an *optimization algorithm* on a single problem instance π. Since optimization algorithms are, in many cases, randomized, their performance Y on one instance is a random variable that might be described by a probability density/mass function $p(y|\pi)$.

Most commonly, we aim at drawing conclusions about a certain *class* or *population* of instances Π. In this case, the performance Y of the algorithm on the class Π is described by the probability function

$$p(y) = \sum_{\pi \in \Pi} p(y|\pi)p(\pi), \tag{10.1}$$

with $p(\pi)$ being the probability of sampling instance π. In other terms, we are interested in the distribution of Y marginalized over the population of instances. This modeling approach is described also by McGeoch (1996), Wolpert and Macready (1997), and Birattari (2004).

In experiments, we sample the population of instances and on each sampled instance we collect sample data on the performance of the algorithm. If on an instance π we run the algorithm r times then we have r replicates of the performance measure Y, denoted by $Y_1, \ldots, Y_r$, which are, conditionally on the sampled instance and given the random nature of the algorithm, independent and identically distributed (i.i.d.), i.e.,

$$p(y_1,\dots,y_r|\pi) = \prod_{j=1}^{r} p(y_j|\pi). \tag{10.2}$$

Marginally (over all the instances) the observed performance measures may show dependence, as is seen from

$$p(y_1,\dots,y_r) = \sum_{\pi\in\Pi} p(y_1,\dots,y_r|\pi)p(\pi). \tag{10.3}$$

The model (10.3) can be easily extended to the case where several algorithms are applied to the same instance by incorporating fixed effects in the conditional structure of (10.2). Next, we illustrate how this leads naturally to a mixed model.

We organize our presentation in different cases according to the number and type of factors involved. For the sake of conciseness, we identify the cases with the following notation:

$$\left\langle \begin{matrix}\text{algorithm}\\ \text{factors}\end{matrix}, \begin{matrix}\text{number of}\\ \text{instances}\end{matrix} \left(\begin{matrix}\text{instance}\\ \text{factors}\end{matrix}\right), \begin{matrix}\text{number of}\\ \text{runs}\end{matrix} \right\rangle.$$

For example, $\langle N, q(M), r\rangle$ means that we are in the presence of an experimental design with N algorithmic factors, q instances sampled from each combination of M instance factors and r runs of the algorithm per instance. We use lower-case letters when referring to the number of factors and upper-case letters when referring to the number of levels. We indicate the absence of fixed factors by a dash. The round parenthesis indicates nesting and its meaning is better explained in Sect. 10.2.3.

10.2.1 Case $\langle$-, q(-), $r\rangle$: Random-Effects Design

We start with the simplest experiment where one algorithm is evaluated on q instances randomly sampled from a class Π. The experiment is performed as follows. In a first stage an instance is randomly drawn from a population of instances, whereafter the single algorithm is run r times on the instance. Given the stochastic nature of the algorithm, this produces, *conditionally on the sampled instance*, r replications of the performance measure that are i.i.d. We use Y_{ij} to denote the random performance measure obtained in the jth replication of the algorithm on the ith instance.

The instances included in the study are randomly drawn from some population of instances, and the interest is in inferring about this larger population of instances, not just on those included in the experiment. The above considerations lead us to propose the following random-effects model:

$$Y_{ij} = \mu + \tau_i + \varepsilon_{ij}, \tag{10.4}$$

where

μ is an overall mean,
τ_i is a random variable representing the effect of instance i, and
ε_{ij} is a random error term for replication j on instance i.

As such, the stochastic behavior of the response variable originates from both the instance and the algorithm. Concerning the random elements in the right-hand side of (10.4) we assume the following:

– $\tau_1, \ldots, \tau_q$ are i.i.d. $N(0, \sigma_\tau^2)$,
– ε_{ij}, $i = 1, \ldots, q$, $j = 1, \ldots, r$, are i.i.d. $N(0, \sigma^2)$,
– all τ_i and ε_{ij} are independent of each other.

Note that the postulated random-effects model satisfies the structure of the conditional and marginal models given by (10.2) and (10.3). In particular, the conditional distribution of the performance measure given the instance is given by

$$Y_{ij}|\tau_i \sim N(\mu + \tau_i, \sigma^2), \quad j = 1, \ldots, r.$$

Furthermore, conditionally on the random effect τ_i, the random variables $Y_{i1}, \ldots, Y_{ir}$ are independent. Integrating out the random effects we obtain the unconditional model

$$Y_{ij} \sim N(\mu, \sigma^2 + \sigma_\tau^2), \qquad i = 1, \ldots, q,\ j = 1, \ldots, r.$$

The use of random instance effects yields dependency between the performance measurements obtained on a specific instance, while performances are independent if they pertain to different instances. Hence, the covariance structure of the model (10.4) is

$$\mathrm{COV}(Y_{ij}, Y_{i'j'}) = \begin{cases} \sigma^2 + \sigma_\tau^2, & \text{if } i = i' \text{ and } j = j', \\ \sigma_\tau^2, & \text{if } i = i' \text{ and } j \neq j', \\ 0, & \text{if } i \neq i', \end{cases} \tag{10.5}$$

which is the compound symmetric covariance structure. The parameters σ^2 and σ_τ^2 determine the variance of the individual Y_{ij} as well as the covariance between the Y_{ij}, and therefore are called the *variance components*.

Collecting the performance measurements $Y_{i1}, \ldots, Y_{ir}$ into the vector $\mathbf{Y}_i$, and denoting by $\mathbf{1}$ the r-dimensional vector of ones and by $\mathbf{\Sigma}$ the $(r \times r)$ covariance matrix of $\mathbf{Y}_i$, i.e.,

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma^2 + \sigma_\tau^2 & \sigma_\tau^2 & \cdots & \sigma_\tau^2 \\ \sigma_\tau^2 & \sigma^2 + \sigma_\tau^2 & \cdots & \sigma_\tau^2 \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_\tau^2 & \sigma_\tau^2 & \cdots & \sigma^2 + \sigma_\tau^2 \end{bmatrix},$$

we can summarize the above as

$$\mathbf{Y}_i \sim N_r(\mu \mathbf{1}, \boldsymbol{\Sigma}), \qquad i = 1, \dots, q,$$

independently.

Note that, in ordinary ANOVA, the *instance* factor in model (10.4) would be considered a fixed factor, i.e., nonrandom, yielding

$$Y_{ij} \sim N(\mu + \tau_i, \sigma^2), \quad i = 1, \dots, q,\ j = 1, \dots, r,$$

with Y_{ij} being independent; i.e., unlike model (10.4), this model does not take into account dependencies arising from applying an algorithm repeatedly to the same instances.

Given that the instances are here considered as samples from some larger population of instances, we are not interested in performing a hypothesis test about the particular levels included in the study. Instead, the interest is in the whole population of instances and hence the hypothesis of interest is one involving the variance component σ_τ^2, in particular

$$H_0 : \sigma_\tau^2 = 0 \qquad \text{versus} \qquad H_1 : \sigma_\tau^2 > 0. \tag{10.6}$$

Clearly, if H_0 is true then the instance distribution reduces to a point mass at zero, implying that all possible instance parameters are fixed and equal to zero, which corresponds to no instance effect.

Intuitively, tests concerning σ_τ^2, as specified in (10.6), and tests involving a comparison of the algorithmic variance σ^2 and the instance variance σ_τ^2 should be based on a comparison of the between-instance variability, measured by $r\sum_{i=1}^q(\bar{Y}_{i.} - \bar{Y}_{..})^2$, and the within-instance variability, measured by $\sum_{i=1}^q \sum_{j=1}^r (Y_{ij} - \bar{Y}_{i.})^2$ (the dot and the bar in $\bar{Y}_{i.}$ indicate averages over the index j and in $\bar{Y}_{..}$ average over both indices i and j). Statistical theory motivates the use of the ratio

$$F = \frac{\frac{r\sum_{i=1}^q(\bar{Y}_{i.}-\bar{Y}_{..})^2}{(q-1)(\sigma^2+r\sigma_\tau^2)}}{\frac{\sum_{i=1}^q\sum_{j=1}^r(Y_{ij}-\bar{Y}_{i.})^2}{q(r-1)\sigma^2}}, \tag{10.7}$$

as it can be shown that under the above model assumptions $F \sim F(q-1, q(r-1))$, where $F(\nu_1, \nu_2)$ is used to denote the F distribution with ν_1 and ν_2 degrees of freedom. We distinguish three specific uses of (10.7):

– Test for an instance effect: $H_0 : \sigma_\tau^2 = 0$ versus $H_1 : \sigma_\tau^2 > 0$.
  Under $H_0 : \sigma_\tau^2 = 0$ we have

$$F_1 = \frac{\text{MSI}}{\text{MSE}},$$

  where

$$\text{MSI} = \frac{r\sum_{i=1}^q(\bar{Y}_{i.} - \bar{Y}_{..})^2}{q-1},$$

$$\text{MSE} = \frac{\sum_{i=1}^{q}\sum_{j=1}^{r}(Y_{ij} - \bar{Y}_{i.})^2}{q(r-1)},$$

and $F_1 \sim F(q-1, q(r-1))$, leading to the decision rule to reject H_0 at the significance level α if $f_1 > F(1-\alpha; q-1, q(r-1))$, where f_1 is the realization of F_1 from the observed data. An intuitive motivation for the form of statistic F_1 can be obtained from the expected mean squares. It can be shown that

$$\text{E[MSI]} = \sigma^2 + r\sigma_\tau^2, \text{ and} \tag{10.8}$$

$$\text{E[MSE]} = \sigma^2, \tag{10.9}$$

so under H_0 both MSI and MSE estimate σ^2 in an unbiased way, and F_1 can be expected to be close to one. On the other hand, large values of F_1 give evidence against H_0.

– Test involving a comparison of the instance and the algorithmic variance: $H_0 : \sigma_\tau^2/\sigma^2 = c$ versus $H_1 : \sigma_\tau^2/\sigma^2 \neq c$.
Under $H_0 : \sigma_\tau^2/\sigma^2 = c$ we have

$$F_2 = \frac{\frac{r\sum_{i=1}^{q}(\bar{Y}_{i.} - \bar{Y}_{..})^2}{(q-1)(1+rc)}}{\frac{\sum_{i=1}^{q}\sum_{j=1}^{r}(Y_{ij} - \bar{Y}_{i.})^2}{q(r-1)}} \sim F(q-1, q(r-1)),$$

leading to the decision rule to reject H_0 if $f_2 < F(\alpha/2; q-1, q(r-1))$ or $f_2 > F(1-\alpha/2; q-1, q(r-1))$.

– Power calculations. Power calculations can be useful at the design stage of the experiment when one has to decide on the number of instances q and the number of replicates r. The power of a statistical test is the probability that the test will reject the null hypothesis when in fact the alternative hypothesis is true. The power of the F test for testing $H_0 : \sigma_\tau^2 = 0$ vs $H_1 : \sigma_\tau^2 > 0$ can be computed from

$$\text{POWER} = \Pr\left\{F > \frac{F(1-\alpha; q-1, q(r-1))}{1 + r\tilde{\sigma}_\tau^2/\sigma^2}\right\}, \tag{10.10}$$

where $\tilde{\sigma}_\tau^2$ is a value for σ_τ^2 from H_1. We refer to Sect. 10.4.1 for an illustration of the use of power calculations.

It might be also relevant to *estimate* the overall mean μ. Since we have $\text{E}[Y_{ij}] = \mu$ then an unbiased estimator of μ is

$$\hat{\mu} = \bar{Y}_{..}.$$

It can be shown that an unbiased estimator[1] of $\sigma^2[\bar{Y}_{..}]$ is $s^2[\bar{Y}_{..}] = \text{MSI}/qr$ and

[1] We adopt the convention of using the same symbol for estimators and estimates, as mentioned in the appendix, pages 427 and 430.

$$\frac{\bar{Y}_{\cdot\cdot} - \mu}{\hat{s}[\bar{Y}_{\cdot\cdot}]} \sim t(q-1).$$

Hence, we obtain the *confidence limits* for μ by

$$\bar{y}_{\cdot\cdot} \pm t(1-\alpha/2; q-1)s[\bar{Y}_{\cdot\cdot}]. \tag{10.11}$$

Confidence intervals on the variance components σ_τ^2 and σ^2 can also be derived, see Kutner et al. (2005).

10.2.2 Case $\langle N, q(\text{-}), r\rangle$: Mixed-Effects Design

We now consider the case where h *algorithms* are evaluated on q *instances* randomly sampled from a class Π. The experiment is performed as follows. In a first stage an instance is sampled from a population of instances. Next, each algorithm is run r times on the instance. Again, *conditionally on the instance* and for a given algorithm, we obtain r i.i.d. replications of the performance measure. We use Y_{ijk} to denote the random performance measure obtained in replication k of algorithm j on instance i. Note that we are here, for simplicity of exposition, dealing with a special case of design $\langle N, q(\text{-}), r\rangle$, namely the case where $N = 1$, which corresponds to having one single factor representing the different algorithms.

The algorithms included in the study are the ones in which we are particularly interested, and hence they can be considered as levels of a *fixed factor*. As before, the instances are drawn randomly from some population of instances and the interest is in inferring about this global population of instances, not just those included in the study. Hence, we assume that the performance measure can be decomposed according to the following *mixed-effects ANOVA model*:

$$Y_{ijk} = \mu + \alpha_j + \tau_i + \gamma_{ij} + \varepsilon_{ijk}, \tag{10.12}$$

where

μ	is an overall performance level common to all observations,
α_j	is a fixed effect due to the algorithm j,
τ_i	is a random effect associated with instance i,
γ_{ij}	is a random interaction between instance i and algorithm j,
ε_{ijk}	is a random error for replication k of algorithm j on instance i.

For identification purposes we impose the usual sum constraint on the factor level effects, i.e., $\sum_{j=1}^{h} \alpha_j = 0$. The assumptions imposed on the random elements are

- τ_i are i.i.d. $N(0, \sigma_\tau^2)$,
- γ_{ij} are i.i.d. $N(0, \sigma_\gamma^2)$,
- ε_{ijk} are i.i.d. $N(0, \sigma^2)$, and
- τ_i, γ_{ij} and ε_{ijk} are mutually independent random variables.

Also here the postulated model satisfies the structure of the conditional and marginal models given by (10.2) and (10.3). In particular, the conditional distribution of the performance measure given the instance and the instance–algorithm interaction is given by

$$Y_{ijk}|\tau_i, \gamma_{ij} \sim N(\mu + \alpha_j + \tau_i + \gamma_{ij}, \sigma^2),\ i = 1, \ldots, q,\ j = 1, \ldots, h,\ k = 1, \ldots, r.$$

Furthermore, conditionally on the random effects τ_i and γ_{ij}, $i = 1, \ldots, q$, $j = 1, \ldots, h$, all responses are independent. Integrating out the random effects we obtain the marginal model for the response variables:

$$Y_{ijk} \sim N(\mu + \alpha_j, \sigma^2 + \sigma_\tau^2 + \sigma_\gamma^2), \quad i = 1, \ldots, q,\ j = 1, \ldots, h,\ k = 1, \ldots, r.$$

The use of random instance effects and random instance–algorithm interactions yields dependency between the performance measurements obtained on a specific instance, while observations are independent if they pertain to different instances. The covariance between any two observations under model (10.12) is

$$\mathrm{Cov}(Y_{ijk}, Y_{i'j'k'}) = \begin{cases} \sigma^2 + \sigma_\tau^2 + \sigma_\gamma^2, & \text{if } i = i',\ j = j',\ k = k', \\ \sigma_\tau^2 + \sigma_\gamma^2, & \text{if } i = i',\ j = j',\ k \neq k', \\ \sigma_\tau^2, & \text{if } i = i',\ j \neq j', \\ 0, & \text{if } i \neq i'. \end{cases} \tag{10.13}$$

The mixed model (10.12) with its assumptions forms the natural basis for testing hypotheses about both fixed and random factors, and their interactions. Concerning the fixed factors, the interest is usually in testing whether there is a difference between the factor level means $\mu + \alpha_1, \ldots, \mu + \alpha_h$. Formally, one tests the hypothesis

$$\begin{aligned} H_0 &: \alpha_1 = \alpha_2 = \ldots = \alpha_h = 0, \\ H_1 &: \text{ at least one } \alpha_j \text{ not equal to 0.} \end{aligned}$$

Similarly to the random-effects model, for the random effects, tests about the particular levels included in the study are meaningless. Instead we test hypotheses about the variance components σ_τ^2 and σ_γ^2, reflecting that the ultimate interest is in the whole population of instances:

$$\begin{aligned} H_0 &: \sigma_\tau^2 = 0, \\ H_1 &: \sigma_\tau^2 > 0, \end{aligned} \qquad \text{and} \qquad \begin{aligned} H_0 &: \sigma_\gamma^2 = 0, \\ H_1 &: \sigma_\gamma^2 > 0, \end{aligned}$$

respectively. In balanced designs, the test statistics for these hypotheses are ratios of mean squares that are chosen such that the expected mean squares of the numerator differs from the expected mean squares of the denominator only by the variance components of the random factor in which we are interested. We report the resulting analysis of variance in Table 10.1. Formal procedures that automatize the derivation of these tables are described by Montgomery (2005, p. 502), Kutner et al. (2005), and Molenberghs and Verbeke (1997, 2005).

Effects	Mean squares	df	Expected mean squares	Test statistics
Fixed factor	MSF	$h-1$	$\sigma^2 + r\sigma^2_\gamma + rp\frac{\sum_{j=1}^h \alpha_j^2}{h-1}$	MSF/MSFR
Random factor	MSR	$p-1$	$\sigma^2 + r\sigma^2_\gamma + rh\sigma^2_\tau$	MSR/MSFR
Interaction	MSFR	$(h-1)(p-1)$	$\sigma^2 + r\sigma^2_\gamma$	MSFR/MSE
Error	MSE	$hq(r-1)$	σ^2	

Table 10.1: Expected mean squares and consequent appropriate test statistics for a mixed two-factor model. For a generalization to multifactorial cases see rules in Montgomery (2005), and Kutner et al. (2005)

Estimators for the fixed effects of balanced mixed models have also been derived. Estimators of the fixed effects α_j are $\hat{\alpha}_j = \bar{Y}_{\cdot j\cdot} - \bar{Y}_{\ldots}$. Perhaps more interesting for our purposes in the analysis of optimization algorithms is the marginal mean $\mu_{\cdot j} = \mu + \alpha_j$ whose best estimator is $\hat{\mu}_{\cdot j} = \bar{Y}_{\ldots} + (\bar{Y}_{\cdot j\cdot} - \bar{Y}_{\ldots}) = \bar{Y}_{\cdot j\cdot}$. Unbiased estimators for the variances $\sigma^2[\hat{\alpha}_j]$ and $\sigma^2[\hat{\mu}_{\cdot j}]$ are

$$s^2[\hat{\alpha}_j] = \frac{1}{qr}\mathrm{MSFR} \quad \text{and} \quad s^2[\hat{\mu}_{\cdot j}] = \frac{h-1}{hqr}\mathrm{MSFR} + \frac{1}{hqr}\mathrm{MSR},$$

respectively. We can then compute exact confidence limits on pairwise comparisons of fixed effects, $D = \mu_{\cdot j} - \mu_{\cdot j'} = \alpha_j - \alpha_{j'}$, by the fact that

$$\frac{\hat{D} - D}{s[\hat{D}]} \sim t\big((h-1)(p-1)\big) \ ,$$

where $s^2[\hat{D}] = 2s^2[\hat{\alpha}_j]$.

Tukey's multiple comparison procedure can be used to guarantee a *family confidence coefficient* $1-\alpha$ when multiple comparisons are to be performed. In other terms, if we perform $\binom{h}{2}$ pairwise comparisons for the fixed factor, we want each of them to be correct $(1-\alpha)100$ percent of times. Tukey's procedure consists of substituting the t distribution with the Studentized range distribution. More precisely, we can make explicit the multiple comparisons confidence limits for all pairwise comparisons $D = \mu_{\cdot j} - \mu_{\cdot j'}$, or equivalently $D = \alpha_j - \alpha_{j'}$, with family confidence coefficient of at least $1-\alpha$ as follows:

$$\hat{D} \pm Ts[\hat{D}], \qquad T = \frac{1}{\sqrt{2}}t'(1-\alpha; h, (h-1)(q-1)).$$

where $t'(p;\nu_1,\nu_2)$ denotes quantile p of the Studentized range distribution with ν_1 and ν_2 degrees of freedom.

A *paired comparison plot* (see Fig. 10.5 in Sect. 10.4.2) can be used to visualize Tukey's multiple comparisons when the design is perfectly balanced. It consists of plotting around each estimated mean, e.g., $\hat{\mu}_{\cdot j}$, an interval whose limits are $\bar{y}_{\cdot j} \pm$

$Ts[\hat{D}]/2$. When intervals overlap in this plot we conclude that there is not significant difference between the means compared. The advantage of this plot is that it shows at the same time the significance and the entity of the differences.

10.2.3 Case $\langle$-, $q(M), r\rangle$: Nested-Effects Design

In the previously considered designs the instances were assumed to be sampled from some (homogeneous) population of instances, whereafter they were solved by all algorithms. Each instance was combined with all possible levels of the algorithmic factors and hence the instance factor and the algorithmic factors were crossed. It is clear that, besides the algorithmic factors, also the characteristics of the instances may affect the performance measure, and that to study these formally we have to include them as fixed factors in the experimental design. Assume that the instances can be characterized by M factors, each of them having a given number of levels. The combination of the levels of these factors defines an instance class and the instances are then randomly sampled from it. As such, instances are specific for the given combination of levels of the instance factors, meaning that they are not crossed with the instance factors, but nested within them.

We consider the simplest design, where there is only one instance factor with m possible levels, defining m instance classes. From each instance class (factor level), q instances are randomly sampled and subsequently solved r times by a single algorithm. We denote here by Y_{ijk} the random performance measure obtained in the kth replication of the algorithm on the ith instance sampled from the jth class. Hereafter, we use the subscript $i(j)$ to indicate that the ith factor level of the random factor is nested within the jth factor level of the fixed factor. A possible model for this design is

$$Y_{ijk} = \mu + \beta_j + \tau_{i(j)} + \epsilon_{ijk}, \qquad (10.14)$$

where

μ	is an overall performance level common to all observations,
β_j	is a fixed effect due to the instance class j,
$\tau_{i(j)}$	is a random effect associated with instance i sampled from class j, and
ε_{ijk}	is a random error for replication k on instance i in class j.

The assumptions on the random effects are as follows:

- $\tau_{i(j)}$ are i.i.d. $N(0, \sigma_\tau^2)$,
- ε_{ijk} are i.i.d. $N(0, \sigma^2)$,
- $\tau_{i(j)}$ and ε_{ijk} are independent random variables.

The principle of nesting is illustrated in Fig. 10.1. Under the above model the response variables are linear combinations of independent normal random variables and hence they follow a normal distribution. To be specific

Class 1			**Class 2**		
Instance			Instance		
1	2	3	1	2	3
Y_{111}	Y_{211}	Y_{311}	Y_{121}	Y_{221}	Y_{321}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Y_{11r}	Y_{21r}	Y_{31r}	Y_{12r}	Y_{22r}	Y_{32r}

Fig. 10.1: An illustration of nested factor design with $m = 2$ and $q = 3$. The instances within the two classes are different and should be more appropriately identified by $1, 2, 3$ and $4, 5, 6$

Effects	Square mean	df	Expected square mean	Test statistics
Fixed factor	MSF	$m-1$	$\sigma^2 + r\sigma_\tau^2 + rp\frac{\sum_{j=1}^m \beta_j^2}{m-1}$	$\mathrm{MSF}/\mathrm{MSI}(F)$
Nested factor	$\mathrm{MSI}(F)$	$m(p-1)$	$\sigma^2 + r\sigma_\tau^2$	$\mathrm{MSI}(F)/\mathrm{MSE}$
Error	MSE	$mq(r-1)$	σ^2	

Table 10.2: Expected mean squares and consequent appropriate test statistics for a two-factor nested model. For a generalization to multifactorial cases see the rules in Montgomery (2005), and Kutner et al. (2005)

$$Y_{ijk} \sim N(\mu + \beta_j, \sigma^2 + \sigma_\tau^2), \quad i = 1, \ldots, q, \ j = 1, \ldots, m, \ k = 1, \ldots, r.$$

The above model forms the basis for performing inference about the factor effects β_j. On the other hand the instances are randomly drawn from some larger population of instances and we focus on the variability by testing the variance component σ_τ^2 of the instances similarly to the previous two cases. The quantities needed for developing the tests, and the test statistics themselves, are presented in Table 10.2.

10.2.4 Case $\langle N, q(M), r\rangle$: General Mixed-Effects Design

In this case, the researcher wishes to assess how the performance measure Y is affected by several parameters of the algorithms and of the instances. Ideally, we fix those parameters that are most important and that we can control, and randomize those properties that we do not understand or cannot control. The parameters controlled may be both categorical or numerical. We consider the following setting:

- Factors $A_1, \ldots, A_N$ represent the parameters of the algorithms. Each combination of these factors gives rise to an instantiated algorithm.
- Factors $B_1, \ldots, B_M$ represent the parameters of the instances (or the *stratification factors* of the whole space of instances). Each combination of these factors gives rise to a different class of instances Π_l.

- From each class of instances Π_l, q instances are sampled randomly and on each of them, each instantiated algorithm is run r times.

The factors A_i, $i = 1, \ldots, N$, and B_j, $j = 1, \ldots, M$, are fixed factors and the factor instance is a random factor. Since the instances within each class Π_l are different, the design is *nested*. This yields a linear mixed model that can be written as

$$\begin{aligned} Y_{i_1,\ldots,i_N,j_1,\ldots,j_M,k} = \mu + \alpha_{i_1} + \ldots + \alpha_{i_N} + \beta_{j_1} + \ldots + \beta_{j_M} + \\ + \tau_{k(j_1,\ldots,j_M)} + \varepsilon_{i_1,\ldots,i_N,j_1,\ldots,j_M,k} \end{aligned} \tag{10.15}$$

with

$\{i_1, \ldots, i_N\}$	an index set for the levels of the algorithmic factors $A_1, \ldots, A_M$;
$\{j_1, \ldots, j_M\}$	an index set for the levels of the instance factors $B_1, \ldots, B_N$;
$\alpha_{i_1}, \ldots, \alpha_{i_N}$	the main effects of the algorithmic factors $A_1, \ldots, A_M$;
$\beta_{j_1}, \ldots, \beta_{j_M}$	the main effects of the instance factors $B_1, \ldots, B_M$;
$\tau_{k(j_1,\ldots,j_M)}$	the random effect of instance k in setting $\{j_1, \ldots, j_M\}$ of the instance factors;
$\varepsilon_{i_1,\ldots,i_N,j_1,\ldots,j_M,k}$	a random error term.

and where, for brevity, we omitted the interaction terms between all fixed factors.

The analysis of this model is a generalization of those outlined in the previous cases. However, it is more convenient to cast the current model into the framework of the linear mixed model (LMM), where (10.15) can be rewritten as

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\mathbf{b} + \boldsymbol{\varepsilon} \tag{10.16}$$

with

$\mathbf{Y}$	an n-vector that contains the response variables,
$\mathbf{X}$	a known $n \times k$ matrix, the design matrix associated with the fixed regression coefficients,
$\boldsymbol{\beta}$	a k-vector that contains the fixed regression coefficients,
$\mathbf{Z}$	a known $n \times q$ matrix, the design matrix associated with the random regression coefficients,
$\mathbf{b}$	an q-vector that contains the random regression coefficients,
$\boldsymbol{\varepsilon}$	a n-vector of error terms.

The terms $\mu + \alpha_{i_1} + \ldots + \alpha_{i_N} + \beta_{j_1} + \ldots + \beta_{j_M}$ for all combinations of indices are now represented by $\mathbf{X}\boldsymbol{\beta}$ and the model is more general because it allows the inclusion of both qualitative and quantitative variables while in the models encountered above variables were limited to be qualitative. Model (10.16) contains two random components, namely $\mathbf{b}$ and $\boldsymbol{\varepsilon}$, for which we make the following distributional assumptions:

$$\mathbf{b} \sim N_q(\mathbf{0}, \mathbf{D}) \qquad \text{and} \qquad \boldsymbol{\varepsilon} \sim N_n(\mathbf{0}, \boldsymbol{\Sigma}),$$

with $\mathbf{b}$ and ε independent random vectors, and where $N_\ell(\mu, \boldsymbol{\Psi})$ denotes the ℓ-variate normal distribution with mean vector μ and covariance matrix $\boldsymbol{\Psi}$. Clearly, the LMM satisfies the conditional and marginal structures outlined in the introduction of Sect. 10.2; in particular, conditionally,

$$\mathbf{Y}|\mathbf{b} \sim N_n(\mathbf{X}\beta + \mathbf{Z}\mathbf{b}, \boldsymbol{\Sigma}) \tag{10.17}$$

and, marginally,

$$\mathbf{Y} \sim N_n(\mathbf{X}\beta, \mathbf{V}[\alpha]), \tag{10.18}$$

where $\mathbf{V}[\alpha] = \mathbf{Z}\mathbf{D}\mathbf{Z}' + \boldsymbol{\Sigma}$. We use the notation $\mathbf{V}[\alpha]$ here in order to indicate explicitly the dependence of the marginal covariance matrix on an unknown vector α of parameters in the covariance matrices $\mathbf{D}$ and $\boldsymbol{\Sigma}$. Although several methods are available to estimate the LMM, the classical approach is based on maximum likelihood estimation of the marginal model (10.18). According to the latter, one maximizes the likelihood function given by

$$L(\theta) = \frac{1}{(2\pi)^{n/2}|\mathbf{V}[\alpha]|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{Y} - \mathbf{X}\beta)'\mathbf{V}^{-1}[\alpha](\mathbf{Y} - \mathbf{X}\beta)\right],$$

with respect to the vector $\theta = (\beta, \alpha)$ of unknown model parameters, leading to the maximum likelihood estimator (MLE) $\widehat{\theta} = (\widehat{\beta}, \widehat{\alpha})$. For a given fixed α, the MLE for β can be obtained explicitly, and is given by

$$\widehat{\beta} = (\mathbf{X}'\mathbf{V}^{-1}[\alpha]\mathbf{X})^{-1}\mathbf{X}'\mathbf{V}^{-1}[\alpha]\mathbf{Y},$$

the well-known generalized least squares estimator for the marginal model (10.18). However, in practice α is typically unknown and hence it must be replaced by an estimate. For this one often uses the restricted maximum likelihood (REML) method, which allows one to estimate α without having to estimate the parameters of β first. The basic idea of the REML method is to form linear combinations $\mathbf{K}'\mathbf{Y}$, where $\mathbf{K}$ is a matrix of full column rank, such that the joint distribution of these transformed data no longer depends on β. This is achieved by constructing $\mathbf{K}$ having columns orthogonal to the columns of $\mathbf{X}$, i.e., $\mathbf{K}'\mathbf{X} = \mathbf{0}$. Another motivation for the REML method stems from the fact that it produces estimates that are less biased than the MLEs.

An appealing feature of the likelihood framework is that it provides a general procedure for testing hypotheses about model parameters, by simply comparing two likelihood values: the likelihood of a restricted model, the null model, and that of an unrestricted model, also referred to as the full model. This approach to hypothesis testing is especially useful in complex and unbalanced designs, where exact tests such as the F tests described above are typically unavailable. Formally, consider the above-introduced parameter vector θ and its associated parameter space $\boldsymbol{\Theta}$ (the set of possible values for θ), so $\theta \in \boldsymbol{\Theta}$, and let θ_1 denote the subvector of θ that is of interest for testing. In other terms, let the vector θ_1 contain the parameters of the unrestricted model that are not contained in the restricted model. Under the null hypothesis $H_0 : \theta_1 = \theta_{10}$, the parameter vector θ is restricted to lie in some subset $\boldsymbol{\Theta}_0$ of $\boldsymbol{\Theta}$. To test the hypothesis about θ_1 one computes the likelihood ratio test statistic expressed by the ratio between the maximum likelihood of the sample data under the restricted model and the one under the unrestricted model, i.e.,

$$\Lambda = \frac{\max_{\theta \in \boldsymbol{\Theta}_0} L(\theta)}{\max_{\theta \in \boldsymbol{\Theta}} L(\theta)},$$

and rejects H_0 if Λ is too small. It can be shown that, under certain regularity conditions and assuming that $H_0 : \theta_1 = \theta_{10}$ holds, $-2 \ln \Lambda$ is approximately distributed as χ^2_ν, provided the sample size is large. The degrees of freedom ν of the approximating chi-squared distribution are given by the difference of the dimensions of the parameter spaces $\boldsymbol{\Theta}_0$ and $\boldsymbol{\Theta}$. One of the assumptions for having a chi-squared limiting distribution for $-2 \ln \Lambda$ is that the parameters in the null hypothesis are not on the boundary of the parameter space. In the LMM we are often interested in testing a hypothesis about a random effect that takes the form $H_0 : \sigma^2 = 0$, which constitutes a violation of this assumption. If one uses in such a case the chi-squared approximation with its usual degrees of freedom, then the test will be conservative. For a detailed description of the asymptotic properties of the maximum likelihood method and the likelihood ratio test statistic we refer the reader to Lehmann (2003) and Lehmann and Romano (2008).

10.3 An Application Example in Optimization Heuristic Design

In this section we briefly describe the application example that we will develop in the next section. The example is extracted from a study on heuristic and exact algorithms for the so-called E1-2AUG problem (Bang-Jensen et al. 2009). Here, we focus on an intermediate result of that work concerning local search algorithms. We first describe the problem and, then, sketch the local search schemes from which the algorithms are derived; finally we introduce the test instances.

10.3.1 Definitions and Problem Formulation

In graph theory terminology (see, for example, Bondy and Murty 2008), an edge uv in a connected graph $G = (V, E)$ is a bridge if we can partition V into two sets $S, V - S$ so that uv is the only edge from E with endpoints in both S and $V - S$. A graph is *2-edge-connected* if it is connected and has no bridges.

The 2-edge-connectivity augmentation (E1-2AUG) problem asks for a given undirected 2-edge-connected graph $G = (V, E)$, a fixed spanning connected subgraph of G, $S = (V, F)$, and a nonnegative weight function ω on $E' = E - F$, to find a subset X of E' of minimal total weight so that $A(G) = (V, F \cup X)$ is 2-edge-connected.

We restrict ourselves to only the cases where the graph G is a simple graph and S is a tree. An edge $uv \in E$ which is not in S is said to *cover* those edges of S which correspond to the unique uv-path P_{uv} in S. We assume that every edge uv in F is covered by at least two edges in E'. We call a subset X of E' a *proper augmentation* of S if $A(G) = (V, F \cup X)$ is 2-edge-connected.

Every optimal augmentation X is minimal, that is, no edge can be deleted from X without leaving at least one edge of S uncovered. If a given augmentation is not minimal it can be made so by means of a *trimming* procedure that removes edges from X without leaving any edge of S uncovered.

It can be shown that the E1-2AUG problem is a special case of the general set covering problem (Conforti et al. 2004). That is, the minimal weight augmentation corresponds to the minimal weight selection of edges such that every edge of F is covered by at least an edge from E'.

10.3.2 Local Search Algorithms

Three construction heuristics named lightest addition (`la`), shortest path (`sp`), and greedy covering (`gc`) have been designed by Bang-Jensen et al. (2009). To improve the solution provided by these heuristics, three local search schemes are used. They are based on a first improvement strategy and on three different neighborhood structures.

Addition neighborhood (addn) Neighboring augmentations are obtained by adding k edges from $E' - X$ and trimming the resulting augmentation.

Destruct–reconstruct neighborhood (gcn) Neighboring augmentations are obtained by removing k edges from the current augmentation and reconstructing the resulting improper augmentation by means of the greedy set covering heuristic by Chvatal (Cormen et al. 2001, p. 1035).

Shortest path neighborhood (spn) It consists of deleting k edges and finding the shortest path between pairs of their ending vertices in a suitable digraph. The digraph is constructed considering edges available for the augmentation and not allowing to reinsert deleted edges. After the insertion of the new edges the augmentation is trimmed to make it again minimal.

Our task is to assess empirically the impact of three factors: the construction heuristic, the local search scheme identified by its neighborhood, and the parameter k common to all neighborhoods.

10.3.3 Problem Instances

In the experiments, we sample the space of instances of the E1-2AUG problem by restricting ourselves to only a portion of it and by stratifying this portion according to three instance characteristics: *type* of graphs, *edge density*, and *distribution of weights*.[2] The type distinguishes between uniform graphs (type `U`), geometric graphs (type `G`) and small world graphs (type `sm`) (see Bang-Jensen et al. 2009 for definitions). In all types of graphs, the spanning tree S is chosen randomly. All graphs may have random weights on their edges (`r`) or have uniform weights (`1`). The edge density is a measure of the amount of edges present in the graph and we consider three possibilities: high, medium, and low, {`h,m,l`}.

10.4 Numerical Examples

We measure the performance of a run of an algorithm on an instance π by the *gap* or *percent error* of the lower bound approximation, i.e., $(z(\pi) - z^*(\pi))/z^*(\pi) \cdot 100$, where $z(\pi)$ is the observed solution cost in that run of the algorithm and $z^*(\pi)$ is the lower bound on the solution costs for that instance. This measure is feasible in our example problem because a good lower bound can be determined for several instances in relatively short time by integer programming. In fact, for most of the cases the lower bound used is also proved to be the optimal solution. Other measures of solution quality are possible. Zemel (1981) points out that a criterion for judging quality measures is the invariance to simple transformation of the instances. Another measure of interest, not based on solution quality, might be the computation time, since the algorithms in the study all have a natural termination condition (the attainment of a local optimum).[3]

We now develop the analysis on the local search algorithms for the E1-2AUG problem proceeding case by case in the same order as in Sect. 10.2. The analysis is conducted with the statistical package R (R Development Core Team 2008) and in the text we give the main commands to execute this analysis. In the online compendium `http://www.imada.sdu.dk/~marco/Mixed/` we report the data, the full code in R, and the same analysis in the statistical software package SAS.

[2] Note that the process of sampling should be designed carefully in order to avoid pitfalls such as bias towards some instances rather than others. For example, the possible nonisomorphic graphs of size 800 are more than those of size 200, hence they should be given more probability to appear. This problem is solved if stratification is applied and the stratifying factor, in the example the size of the graph, is included in the analysis.

[3] Often local search algorithms are enhanced by metaheuristics (Glover and Kochenberger 2002) and no longer have a natural termination condition. In this case, computation time can be seen as an external parameter and treated as an algorithmic fixed factor in the models discussed here.

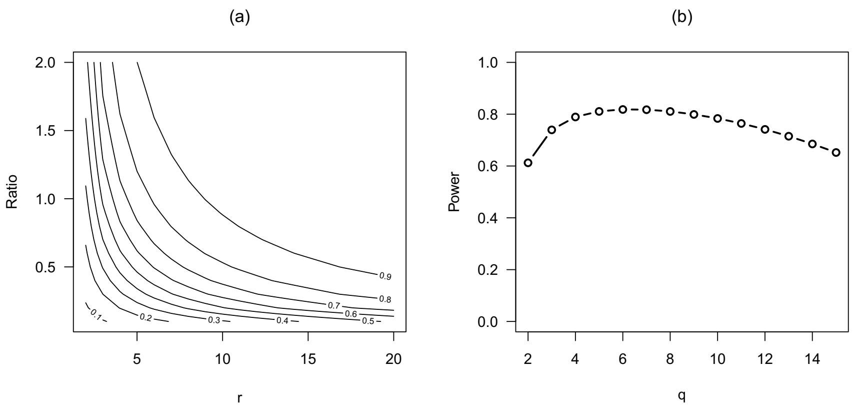


Fig. 10.2: Statistical power for the case $\langle -, q(-), r\rangle$. The power is a function of three variables: $\tilde{\sigma}^2_\tau/\sigma^2$, q, and r. The plot on the *left* shows a contour plot of the power surface as a function of r and $\tilde{\sigma}^2_\tau/\sigma^2$ when $q = 5$. The plot on the *right* shows the power as a function of q for a total number of experiments $qr = 30$, when $\tilde{\sigma}^2_\tau/\sigma^2 = 1$

10.4.1 Case $\langle$-, q(-), $r\rangle$: Random-Effects Design

The goal of this simple case is to illustrate the decomposition of the variance of the response observations in two components, namely, the variability of the results due to the stochasticity of the algorithm and the variability due to the instances sampled from the population. Moreover, we derive an estimation of solution quality with associated confidence intervals.

We collect the results of one algorithm run a number of times on a set of instances. Precisely, the algorithm is determined by the choices `gc`, `addn`, and `k3` and the instance class by the choices `G`, `m`, and `1`.

In the design of the experiment, we decide the number of runs and the number of instances on the basis of considerations on the level of significance and on the power. In particular, we fix the level of significance at 0.05 and aim to a statistical power of 0.8. (These values are maintained throughout the remainder of the chapter.) We then use (10.10) to compute the value of power as a function of r, q, and $\tilde{\sigma}^2_\tau/\sigma^2$ and we visualize this function in two alternative ways in Fig. 10.2.

The two plots represent different views of $\mathrm{POWER}(\tilde{\sigma}^2_\tau/\sigma^2, q, r)$. In Fig. 10.2(a) we show the contour plot of the POWER surface when considered as a function of $\tilde{\sigma}^2_\tau/\sigma^2$ (called ratio) and r, for a fixed value of q (here $q = 5$). Each curve in this plot corresponds to a specific power level and represents the $(r, \tilde{\sigma}^2_\tau/\sigma^2)$ combinations for which this power is achieved. For instance, if one wants to detect with a 5% significance test a $\tilde{\sigma}^2_\tau$ which is of the same magnitude as σ^2 (corresponding to ratio $= 1$) with a probability of 0.8 in an experiment with five instances, then one has to collect six replicates to achieve this level. Fig. 10.2(b) contains an alternative representation of the POWER function. Here we show the power of the test when

$\tilde{\sigma}_\tau^2/\sigma^2 = 1$ as a function of the number of instances q, when the total number of experiments is fixed at 30, i.e., when $qr = 30$. This bound on the number of experiments can be posed by consideration on the computational time available. This plot has a peak around $q = 6$, hinting at a POWER-optimal design under the conditions stated. We use this observation to conclude that, in order to have a power of 0.8 when $\tilde{\sigma}_\tau^2/\sigma^2 = 1$, which we deem a relevant value for practical purposes, then we need to collect at least $r = 30/q = 5$ runs on $q = 6$ instances.[4]

In a second step, we analyze the results of an experiment in which six instances were randomly sampled from the class `G-m-1`, whereafter they were solved five times with the algorithm `gc-addn-k3`. We load the data in R stored in a data frame and check its content by means of the command `str`. The data frame is organized in two columns, `instance` and `algorithm`, that indicate the factors of each observation, a column `run` that reports the replicate number, and a column `gap` that gives the response variable:

```
> load("Data/OPOR.dataR")
> str(OPOR, strict.width = "cut", width = 70)
'data.frame':        30 obs. of  4 variables:
 $ instance : Factor w/ 6 levels "G-800-0.5-1-pre.ins",..: 1 1..
 $ run      : int  5 1 3 2 4 5 3 4 2 1..
 $ algorithm: Factor w/ 1 level "gc-addn-3": 1 1 1 1 1 1 1..
 $ gap      : num  4.07 4.05 4.07 4.07 4.07 ...
```

Before presenting the results of the analysis and performing hypothesis testing using the random-effects model described in Sect. 10.2.1, we comment on the validity of the model assumptions. Under model (10.4) the response variables are normally distributed with mean μ and variance $\sigma^2 + \sigma_\tau^2$, and we validate this assumption using a normal quantile plot (QQ plot). In such a plot we compare the empirical quantiles (the ordered data) with the corresponding quantiles of a standard normal model. In case the normality assumption holds then the points on the QQ plot will show a straight line pattern (see also the appendix of this book). For the experiment under consideration we show this plot in Fig. 10.3(a). Clearly, the points are quite tightly concentrated along a straight line, indicating that a normal model is plausible for our data. In Fig. 10.3(b) and (c), we report the quantile plots also for two other cases that we examined. More precisely, the second QQ plot is obtained for a continuous optimization problem, namely the least median of squares, a robust way to estimate parameters in linear regression analysis (Rousseeuw 1984). The problem with this plot is that the tails are very far from being normally distributed, with the tails of the empirical distribution being lighter than normal tails. The third plot is based on a study for the graph coloring problem (Chiarandini 2005). In this case, the major problem is that data, corresponding to the minimal number of colors used, are discrete and distributed among only a few values. The whole methodology developed in this chapter works for continuous objective functions. When data are discrete but, contrary to the case of the third plot, have many possible values, data can still be reasonably approximated by a continuous distribution.

[4] In the next designs we omit the details of power computations. A computer program by Lenth (2006) for these computations is available online.

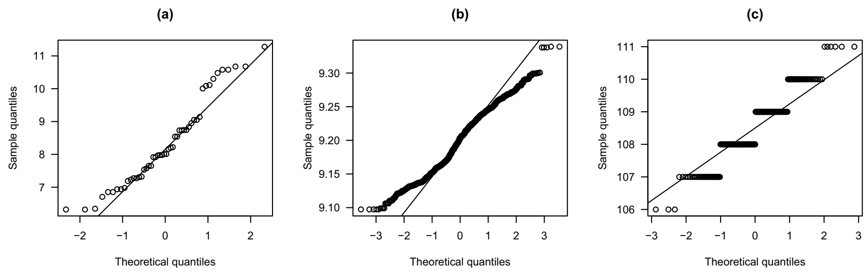


Fig. 10.3: The distribution of data. The leftmost plot shows the quantile distribution of `gc-addn-k3` when run five times on ten instances of the class `G-800-0.5`. The plot in the center shows five runs on 500 instances on a different problem where the distribution shows strong deviance from normality even after data transformation. The rightmost plot shows the distribution of quantiles from five runs on 60 instances. Here the problem is discreteness of data

We now turn to fitting the random-effects model (10.4) and to the estimates and inference about its parameters. Random (and mixed) effects models can be fitted with the function `lmer` of the package `lme4` (Bates et al. 2008).[5] Below we show the resulting R output.

```
> library(lme4)
> fm1a <- lmer(gap ~ 1 + (1 | instance), data = OPOR)
> print(fm1a, digits = 3, corr = FALSE)
```

```
Linear mixed model fit by REML
Formula: gap ~ 1 + (1 | instance)
   Data: OPOR
  AIC   BIC logLik deviance REMLdev
 -101 -96.6   53.4     -105    -107
Random effects:
 Groups   Name        Variance Std.Dev.
 instance (Intercept) 4.362580 2.0887
 Residual             0.000173 0.0131
Number of obs: 30, groups: instance, 6
Fixed effects:
            Estimate Std. Error t value
(Intercept)    4.559      0.853    5.35
```

Since there are no fixed effects, the model (10.4) passed to `lmer` contains only `1` that represents the intercept μ. The random effect is expressed by `(1|instance)`, indicating that the data is grouped by `instance` and that the random effect is constant within each group, `1`. By default `lmer` uses the restricted maximum likelihood (REML) method to fit the model. The output provides information about some of the measures of the fitting such as the log-likelihood (53.4), the deviance for the maximum likelihood criterion (-105), the deviance for the REML criterion (-107), Akaike's information criterion (AIC $= -101$) and Schwartz's Bayesian information criterion (BIC $= -96.6$). Under the header `Fixed effects`, we find the

[5] The package `nlme` (Pinheiro et al. 2008) can also treat mixed-effects models.

estimate for the intercept μ while under `Random effects` we find the estimates for the parameters related to the random effects and the error distributions, here the standard deviations for τ (`instance`) and ε (`Residuals`), respectively. For our experiment we obtain $\hat{\sigma}_\tau = 2.0887$ and $\hat{\sigma} = 0.0131$, which indicates that the variability in the response observations can be mainly attributed to the variability of the instances.

By default the `lmer` function does not report the test on the hypothesis about the variance components σ_τ^2 and σ^2. This is because in general for unbalanced data the computation of the test is not trivial. However, in the cases of perfectly balanced experiments, such as ours, we can proceed to compute the F statistic and the p-value on the basis of (10.8) and (10.9), by plugging the estimates into the equations for the expected mean squares. We get

$$\begin{aligned}\text{MSI} &= \hat{\sigma}^2 + r\hat{\sigma}_\tau^2 \\ &= 0.0131^2 + 5(2.0887)^2, \\ \text{MSE} &= \hat{\sigma}^2 \\ &= 0.0131^2,\end{aligned}$$

and hence $f_1 = 126283$, with p-value ≈ 0 thus the null hypothesis is to be rejected. In R:

```
> VC <- VarCorr(fm1a)
> sigma.tau <- as.numeric(attr(VC$instance, "stddev"))
> sigma <- as.numeric(attr(VC, "sc"))
> q <- nlevels(OPOR$instance)
> r <- length(unique(OPOR$run))
> MSI <- sigma^2 + r * sigma.tau^2
> MSE <- sigma^2
> 1 - pf(MSI/MSE, q - 1, q * (r - 1))
[1] 0
```

We can compute the test on the random effects also by using the likelihood ratio test. In this case, for the likelihood of the model without fixed effects, we have to use the function `lm`

```
> fm1a <- lmer(gap ~ 1 + (1 | instance), data = OPOR, REML = FALSE)
> fm1a.0 <- lm(gap ~ 1, data = OPOR)
> LRT <- as.numeric(2 * (logLik(fm1a) - logLik(fm1a.0)))
> 1 - pchisq(LRT, 1)
[1] 0
```

The test confirms the rejection of the null hypothesis. Note that we perform here a test where the parameter of the null hypothesis is on the boundary of the parameter space, and hence, as noted before, the classical chi-squared approximation to the null distribution of the likelihood ratio test is inappropriate. For this particular case, where we test the importance of a single variance component, the limiting distribution of the likelihood ratio statistic is a mixture of a point mass at zero and a chi-squared distribution with one degree of freedom, where both components of the mixture have probability one. This implies that the usual p-value needs to be divided by two, or, otherwise stated, that the classical test is conservative (Stram and Lee 1994, 1995).

Finally, if we wish to predict the performance of the algorithm on a new instance, the best we can do is to give $\hat{\mu} = 4.559$ and to give the 95% confidence interval. According to (10.11) of Sect. 10.2.1:

```
> s <- sqrt(MSI/(q * r))
> Y.. <- mean(OPOR$gap)
> qsr <- qt(1 - 0.025, 5)
> Y.. - qsr * s
[1] 2.37
> Y.. + qsr * s
[1] 6.75
```

hence $\mu \in [2.37; 6.75]$.

10.4.2 Case $\langle N, q(\text{-}), r\rangle$: Mixed-Effects Design

We discuss two designs within this case: $\langle 1, q(-), r\rangle$ and $\langle N, q(-), r\rangle$. The focus is, in the first design, on the visualization of the results and, in the second design, on the comparison of replicated versus unreplicated designs.

$\langle 1, q(-), r\rangle$ In the first design we aim at comparing the performance of the addition neighborhood at different values of k, over an instance class. More precisely, we have the following factors:

- `algorithm`: three algorithms, starting from the solution produced by greedy covering (`gc`) and using the k-addition neighborhood (`addn`) with $k = \{1, 3, 5\}$, hence levels in `{gc-addn-1,gc-addn-3,gc-addn-5}`
- `instance`: five instances randomly sampled from the class G-m-1
- `replicates`: five

```
> load("Data/YPOR.dataR")
> str(YPOR, strict.width = "cut", width = 70)
'data.frame':        75 obs. of  5 variables:
 $ instance : Factor w/ 5 levels "G-800-0.5-1-pre.ins",..: 1 1..
 $ k        : Factor w/ 3 levels "1","3","5": 3 2 1 3 1 2..
 $ algorithm: Factor w/ 3 levels "gc-addn-1","gc-addn-3",..: 3 2..
 $ run      : int  5 2 4 1 1 1 5 5 4 3 ...
 $ gap      : num  3.67 4.07 4.07 4.05 4.05 ...
```

Relevant questions for this design are:

- Is there an instance effect, i.e., do the instances contribute significantly to the variability of the responses?

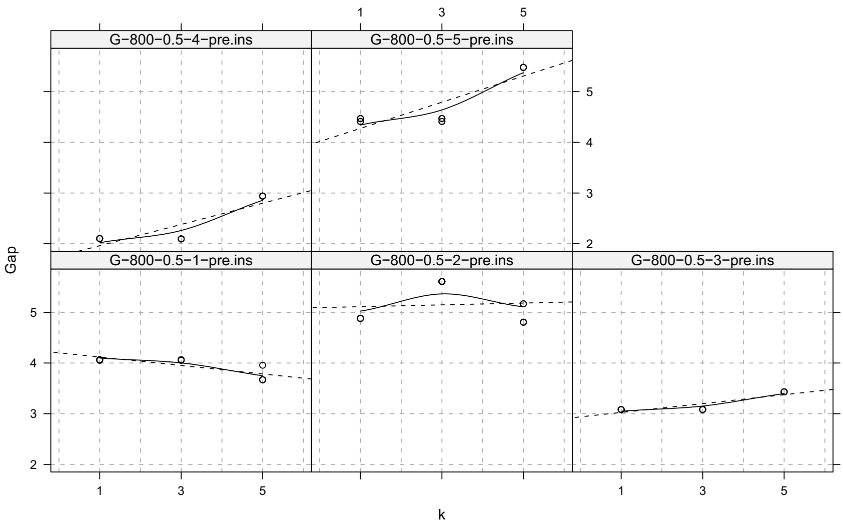


Fig. 10.4: The data in the design $\langle 1, q(-), r\rangle$. The three algorithms, `gc-addn-1`, `gc-addn-3`, and `gc-addn-5`, correspond to greedy covering construction (gc) followed by k-addition neighborhood (addn) with $k = \{1, 3, 5\}$. A local regression line and a least square linear regression line (dashed) are superimposed

- Do the mean performances of the algorithms with different k differ? If yes, how different are they?
- Do the instance–algorithm interactions contribute significantly to the variability of the responses?

We treat k as if it were a qualitative factor, even though k is a numerical value with a clear order. Treating discrete numerical factors as qualitative factors gives more freedom, because in this way we do no assume that the change in the mean response for k from 1 to 3 has to be the same as for k from 3 to 5.

A way to inspect the data is by plotting the percentage error of the algorithms within each instance, which is interpreted as a different group of results. This is shown in Figure 10.4. We observe that there are differences in the slopes and intercepts of the linear regressions within each group. This hints at the presence of random effects and interaction effects between the random and the fixed factors. We will therefore test the inclusion of both a random instance intercept and a random instance–algorithm interaction in the model that describes these data.

The results of the analysis of the mixed model are:

```
> op <- options(contrasts = c("contr.sum", "contr.poly"))
> fm2a <- lmer(gap ~ k + (1 | instance) + (1 | instance:k),
      data = YPOR)
> print(fm2a, digits = 3)
```

```
Linear mixed model fit by REML
Formula: gap ~ k + (1 | instance) + (1 | instance:k)
   Data: YPOR
 AIC   BIC logLik deviance REMLdev
 -95 -81.1   53.5     -111    -107
Random effects:
 Groups     Name        Variance Std.Dev.
 instance:k (Intercept) 0.15795  0.3974
 instance   (Intercept) 1.23514  1.1114
 Residual               0.00387  0.0622
Number of obs: 75, groups: instance:k, 15; instance, 5
Fixed effects:
            Estimate Std. Error t value
(Intercept)   3.8939     0.5075    7.67
k1           -0.1786     0.1455   -1.23
k2           -0.0342     0.1455   -0.23

Correlation of Fixed Effects:
   (Intr) k1
k1  0.000
k2  0.000 -0.500
```

We have specified sum contrasts, here, as a way to identify parameters in the model instead of the default treatment contrasts for `lmer`. This will make later results comparable with `lm`. The estimated variances for the instance and the instance–algorithm interaction random effects are $\hat{\sigma}_\tau^2 = 1.23514$ and $\hat{\sigma}_\gamma^2 = 0.15795$, respectively. The section `Fixed effects` reports the estimates of the fixed effects model parameters from which we obtain the point estimates for the mean performance of the algorithms $\mathrm{E}[Y_{ijk}] = \mu + \alpha_j$. The sum contrasts specified before implies that $\sum \alpha_j = 0$. Hence, for $\alpha_{\mathtt{k1}} = -0.1786$ and $\alpha_{\mathtt{k2}} = -0.0342$, we have $\alpha_{\mathtt{k3}} = 0.2128$, with `k1` representing $k = 1$, `k2`, $k = 3$, and `k3`, $k = 5$. The last column in this section gives the t statistics for the hypotheses that the jth level of the factor is not different from the mean response.

Let us look at the acceptance or rejection of the null hypothesis that the variance components of the random effects are zero. The exact test is via the F-ratio from Table 10.1 of Sect. 10.2.2. If we do not want to look up the table the likelihood ratio test can be computed more easily.

```
> fm2a.1 <- lmer(gap ~ k + (1 | instance), data = YPOR,
     REML = FALSE)
> fm2a.2 <- lmer(gap ~ k + (1 | instance) + (1 | instance:k),
     data = YPOR, REML = FALSE)
> anova(fm2a.2, fm2a.1)
```

```
Data: YPOR
Models:
fm2a.1: gap ~ k + (1 | instance)
fm2a.2: gap ~ k + (1 | instance) + (1 | instance:k)
       Df   AIC   BIC logLik Chisq Chi Df Pr(>Chisq)
fm2a.1  5  71.0  82.6  -30.5
fm2a.2  6 -99.2 -85.3   55.6   172      1     <2e-16
```

As is clear, the instance–algorithm interactions contribute significantly to the variability of the performance measure, and hence, given (10.13), measurements obtained on a particular instance show dependence. A similar test can be performed for the instance variance, by fitting a model without an instance random effect:

```
> fm2a.3 <- lmer(gap ~ k + (1 | instance:k), data = YPOR,
     REML = FALSE)
> anova(fm2a.2, fm2a.3)
```

```
Data: YPOR
Models:
fm2a.3: gap ~ k + (1 | instance:k)
fm2a.2: gap ~ k + (1 | instance) + (1 | instance:k)
       Df   AIC   BIC logLik Chisq Chi Df Pr(>Chisq)
fm2a.3  5 -84.6 -73.0   47.3
fm2a.2  6 -99.2 -85.3   55.6  16.6      1    4.5e-05
```

Hence also this term is significant and should be included in the model.

Let us now analyze the significance of fixed effects. We use the F-ratio[6]

```
> anova(fm2a)
```

```
Analysis of Variance Table
  Df  Sum Sq Mean Sq F value
k  2 0.00956 0.00478    1.23
```

The `lmer` function does not return the p-value for the test on fixed-effects terms, but the F statistic computed by the `anova` function is the correct one for balanced designs. Hence, the observed F statistic is 1.23 on 2 (`Df`) and $(h-1)(q-1)=8$ degrees of freedom and with a p-value of 0.341,

```
> p <- nlevels(YPOR$instance)
> h <- nlevels(YPOR$k)
> r <- length(unique(YPOR$run))
> 1 - pf(anova(fm2a)$"F value", h - 1, (h - 1) * (p - 1))
```

```
[1] 0.341
```

We can conclude that there is not a significant effect of k, and hence that the mean performance measure is not affected by this parameter.

Since we could not reject the global null hypothesis on k, the paired comparison plot will show overlapping confidence intervals for the three values of k. For the sake of completeness in our exposition, we produce this plot (Fig. 10.5, left panel)

```
> VC <- VarCorr(fm2a)
> sigma.gamma <- as.numeric(attr(VC$"instance:k", "stddev"))
> sigma <- as.numeric(attr(VC, "sc"))
> MSIK <- sigma^2 + p * sigma.gamma^2
> Yj. <- with(YPOR, aggregate(gap, list(alg = algorithm),
     mean))
> s <- sqrt(2) * sqrt(MSIK/(p * r))
> T <- qtukey(1 - 0.05, h, (h - 1) * (p - 1))/sqrt(2)
```

[6] Due to implementation issues in R and SAS the likelihood ratio test cannot be used for testing some of the fixed effects, as they remain unidentified (SAS Institute Inc. 2007).

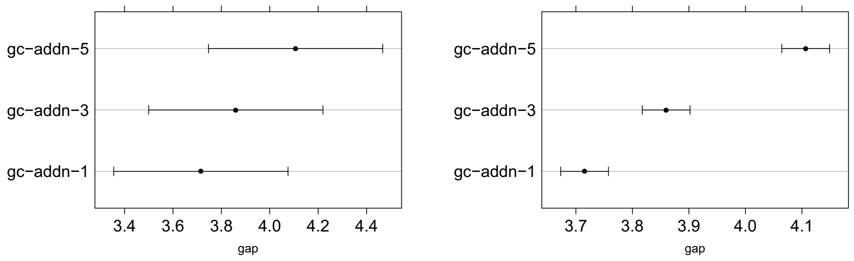


Fig. 10.5: Paired comparison plots. On the left the one obtained by the mixed-effects model, on the right the one obtained by ordinary ANOVA

```
> Yj.$lower <- Yj.$x - 0.5 * T * s
> Yj.$upper <- Yj.$x + 0.5 * T * s
> intervals(alg ~ x, Yj.)
```

The function `intervals` is a wrapper of `dotplot` available from the online compendium at `http://www.imada.sdu.dk/~marco/Mixed/`.

We check the diagnostic plots. We consider the conditional and marginal structure of the model, given in equations (10.17) and (10.18) of Sect. 10.2.4, respectively. In the standard diagnostic plots of residuals against fitted values we check the assumption of homoscedasticity of observations, whereas, in the QQplot we check if residuals meet the assumption of normality. Conditional residuals pertain to each instance individually taken and refer to the distances of observed points from the fitted conditional models. Aggregating these data for the five instances available we see that there might be some deviation from the assumptions, mainly due to the small variability of the responses within an instance. It might then be worth indicating the instances that cause the largest deviation from the assumptions. Things are instead much better for the marginal structure, which is the one of most interest in our study. The plots seem to support quite well the assumptions of homoscedasticity and normality.

```
> plot(fitted(fm2a, type = "response"), residuals(fm2a,
     type = "response"), main = "Conditional residuals",
     xlab = "Predicted", ylab = "Residuals")
> res <- residuals(fm2a, type = "response")
> qqnorm(res, main = "Conditional residuals, QQplot")
> qqline(res)
> fm2a.0 <- lm(gap ~ k, data = YPOR)
> x <- model.matrix(fm2a.0)
> pred <- x %*% fixef(fm2a)
> res <- YPOR$gap - pred
> plot(pred, res, main = "Marginal residuals", xlab = "Predicted",
     ylab = "Residuals")
> qqnorm(res, main = "Marginal residuals, QQplot")
> qqline(res)
```

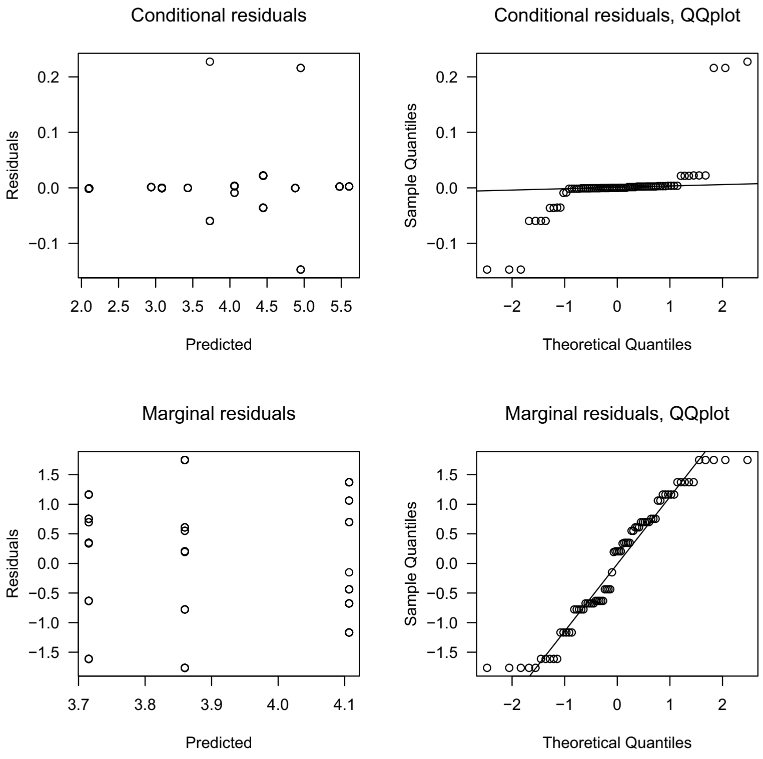


Fig. 10.6: Diagnostic plots

Finally, it is instructive to compare the results obtained here under a random-effects model with those obtained by considering instances as fixed factors. In this latter case, the test for algorithmic differences is performed relative to the mean squared error, and not relative to the instance–algorithm interaction mean squares (Table 10.1, Sect. 10.2.2). Moreover the F test has 60 degrees of freedom at the denominator, compared with 8 under a mixed model, and hence, for the same significance level it would reject sooner.

```
> fm2a.lm <- lm(gap ~ k * instance, data = YPOR)
> anova(fm2a.lm)
```

```
Analysis of Variance Table
Response: gap
            Df  Sum Sq Mean Sq F value Pr(>F)
k            2    2.0     1.0     253 <2e-16
instance     4   77.3    19.3    4986 <2e-16
k:instance   8    6.3     0.8     205 <2e-16
Residuals   60    0.2  0.0039
```

This would have led us to reject the hypothesis on k and conclude, mistakenly, that k is significant! The different conclusion is also shown in Fig. 10.5, where on the right we report the paired comparison plot that would arise from a Tukey pairwise analysis based on the fixed-effect model of lm.

$\langle N, q(-), r\rangle$ We now discuss the case $\langle N, q(-), r\rangle$ and compare replicated and unreplicated designs. This case differs slightly from the previous in that we study three fixed factors and this leads us to a multifactorial analysis. All fixed factors are algorithmic factors and are tested at three levels.

– init.heur: the starting solution generated by three different construction heuristics. It is a categorical factor in the levels {gc, la, sp};
– neigh: the three local search schemes with the neighborhood structures described above. It is a categorical factor in the levels {addn, covn, spn};
– k: the parameter that determines the extension of the neighborhood. It is a categorical factor in the levels {1, 3, 5}.

The 27 possible combinations give rise to 27 algorithms to test. If our computational budget allows us to run 675 experiments then we can choose between a *replicated design* with five instances and five runs per instance, or an *unreplicated design* with one single run of each algorithm on 25 instances.

Let's analyze first the *replicated design*.

```
> load("Data/NPOR.dataR")
> str(NPOR, strict.width = "cut", width = 70)
```

```
'data.frame':        675 obs. of  6 variables:
 $ instance : Factor w/ 5 levels "sm-800-h-w1",..: 1 1 1 1 1..
 $ init.heur: Factor w/ 3 levels "gc","la","sp": 1 1 2 2 3 3..
 $ neigh    : Factor w/ 3 levels "addn","gcn","spn": 2 1 3 1..
 $ k        : Factor w/ 3 levels "1","3","5": 3 2 1 2 1 2 3 2..
 $ run      : int  2 3 1 3 5 5 2 3 3 1 ...
 $ gap      : num  1.521 1.433 0.397 2.976 2.843 ...
```

We test the significance of the random effects and their interactions. The exponent of two in the lmer model statement indicates that all interactions of second order are included.

```
> fm2bR.0 <- lm(gap ~ (k + init.heur + neigh)^2, data = NPOR)
> fm2bR.1 <- lmer(gap ~ (k + init.heur + neigh)^2 + (1 |
     instance), data = NPOR, REML = FALSE)
> fm2bR.2 <- lmer(gap ~ (k + init.heur + neigh)^2 + (1 |
     instance) + (1 | instance:k) + (1 | instance:neigh) +
     (1 | instance:init.heur), data = NPOR, REML = FALSE)
> LRT <- as.numeric(2 * (logLik(fm2bR.2) - logLik(fm2bR.0)))
> 1 - pchisq(LRT, 1)
```

```
[1] 0
```

```
> anova(fm2bR.2, fm2bR.1)
```

```
Data: NPOR
Models:
fm2bR.1: gap ~ (k + init.heur + neigh)^2 + (1 | instance)
fm2bR.2: gap ~ (k + init.heur + neigh)^2 + (1 | instance)
               + (1 | instance:k) + ...
fm2bR.2:     (1 | instance:neigh) + (1 | instance:init.heur)
        Df  AIC  BIC logLik Chisq Chi Df Pr(>Chisq)
fm2bR.1 21 1301 1396   -630
fm2bR.2 24  672  780   -312   636      3     <2e-16
```

The likelihood ratio test indicates again that the random factor instance is significant and also at least one of the random interaction terms between a fixed factor and the instance factor. For the fixed effects we have

```
> fm2bR <- lmer(gap ~ (k + init.heur + neigh)^2 + (1 |
     instance) + (1 | instance:k) + (1 | instance:neigh) +
     (1 | instance:init.heur), data = NPOR)
> anova(fm2bR)
Analysis of Variance Table
                Df Sum Sq Mean Sq F value
k                2    0.4     0.2    1.56
init.heur        2   10.5     5.2   41.65
neigh            2    3.6     1.8   14.36
k:init.heur      4    0.3     0.1    0.69
k:neigh          4   39.6     9.9   78.54
init.heur:neigh  4   37.5     9.4   74.43
```

We omit here the details of the analysis of variance, which is similar to the previous case. It yields a p-value of 0.2675 for `k` and of 0.5984 for the interaction `k:init.heur`, thus leading us to not reject the null hypothesis of no effect for these two factors. The latter result was expected, given that k does not alter the construction heuristics. All other effects are instead significant. To gain insight when interaction terms are significant one can use 2D or 3D interaction plots. In Fig. 10.7 we visualize the interactions `neigh:init.heur` and `k:neigh`.

```
> with(NPOR, {
     interaction.plot(neigh, init.heur, gap, fixed = TRUE)
     interaction.plot(k, neigh, gap, fixed = TRUE)
 })
```

For later comparisons we report also the estimates of the fixed effects:

```
> summary(fm2bR)@coefs[1:10, ]
              Estimate Std. Error t value
(Intercept)     2.3800     0.2509   9.485
k1              0.0340     0.0375   0.909
k2             -0.0662     0.0375  -1.767
init.heur1     -1.0076     0.1118  -9.016
init.heur2      0.3669     0.1118   3.283
neigh1         -1.1496     0.2180  -5.273
neigh2          0.3952     0.2180   1.813
k1:init.heur1   0.0131     0.0273   0.480
k2:init.heur1  -0.0147     0.0273  -0.538
k1:init.heur2  -0.0421     0.0273  -1.542
```

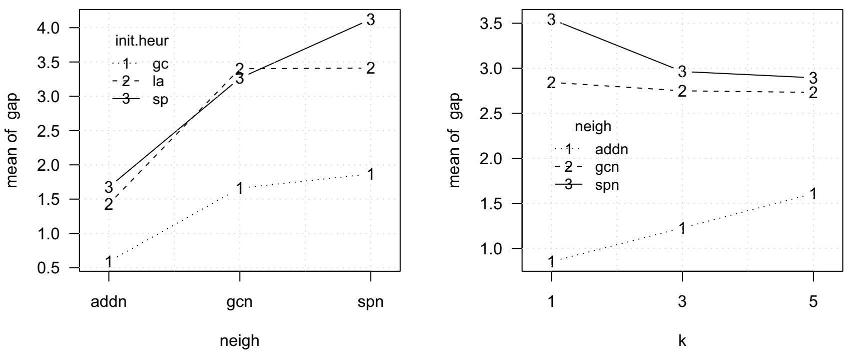


Fig. 10.7: Interaction plots to visualize the impact of `neigh:init.heur` and `k:neigh` in the replicated model `fm2bR`

We now turn to the *unreplicated design*, i.e., $r = 1$.

```
> load("Data/NPOY.dataR")
> str(NPOY, strict.width = "cut", width = 70)
```

```
'data.frame':        675 obs. of  6 variables:
 $ instance : Factor w/ 25 levels "1","10","11",..: 1 1 1 1 1 ..
 $ init.heur: Factor w/ 3 levels "gc","la","sp": 1 1 2 1 3 1 1..
 $ neigh    : Factor w/ 3 levels "addn","gcn","spn": 3 3 2 1 2..
 $ k        : Factor w/ 3 levels "1","3","5": 2 1 3 2 1 3..
 $ run      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ gap      : num  1.565 1.984 1.521 0.397 1.807 ...
```

We omit the likelihood ratio test analysis for the random effects, which is encoded in R exactly in the same way as for the replicated case and yields the same highly significant p-values.

The main point we want to make with this design pertains, instead, to the fixed effects:

```
> fm2bU <- lmer(gap ~ (k + init.heur + neigh)^2 + (1 |
     instance) + (1 | instance:k) + (1 | instance:neigh) +
     (1 | instance:init.heur), data = NPOY)
> anova(fm2bU)
```

```
Analysis of Variance Table
                Df Sum Sq Mean Sq F value
k                2   1.13    0.56    7.62
init.heur        2  10.32    5.16   69.82
neigh            2  10.80    5.40   73.11
k:init.heur      4   0.68    0.17    2.30
k:neigh          4  15.50    3.88   52.47
init.heur:neigh  4  20.07    5.02   67.92
```

Again the only p-values larger than 0.05 are those for `k` and `k:init.heur` (not shown). But the relevant observation is about the estimates of fixed effects means

```
> summary(fm2bU)@coefs[1:10, ]
                Estimate Std. Error t value
(Intercept)       1.7642     0.1487  11.864
k1                0.0423     0.0148   2.861
k2               -0.0552     0.0148  -3.731
init.heur1       -0.7503     0.0639 -11.740
init.heur2        0.3007     0.0639   4.706
neigh1           -0.8072     0.0677 -11.929
neigh2            0.2878     0.0677   4.253
k1:init.heur1     0.0197     0.0209   0.941
k2:init.heur1    -0.0213     0.0209  -1.017
k1:init.heur2    -0.0604     0.0209  -2.889
```

The standard errors of these estimates in the unreplicated case are smaller than those in the replicated case, an observation which is consistent with Birattari (2004). For example, for `k1` we have $s[\alpha_{\texttt{k1}}] = 0.0375$ in the replicated case against $s[\alpha_{\texttt{k1}}] = 0.0148$ in the unreplicated case. As a consequence of this fact, the unreplicated case yields more powerful tests for differences between the levels of the fixed factors. Hence, when a limit on the total number of experiments is imposed, maximizing the number of tested instances should be preferred with respect to maximizing the number of replicates.

10.4.3 Case $\langle$-, $q(M)$, $r\rangle$: Nested Design

In this case we study the effect of instance parameters which are used to stratify the population of instances. We consider only one algorithm, as we are only interested in the instance parameters. Obviously, the conclusions on the instances will be valid only for the algorithm chosen. We have two instance factors under study:

- `type`: the type of graph with levels `{U,G,sm}`
- `weights`: the distribution of weights with levels `{w,1}`;

```
> load("Data/OPMR.dataR")
> str(OPMR, strict.width = "cut", width = 70)
'data.frame':        150 obs. of  8 variables:
 $ weights  : Factor w/ 2 levels "1","w": 1 1 1 1 1 1 1 ...
 $ type     : Factor w/ 3 levels "G","U","sm": 1 1 1 1 1 1 ...
 $ algorithm: Factor w/ 1 level "gc-addn-1": 1 1 1 1 1 1 ...
 $ instance : Factor w/ 30 levels "G-800-1-11","G-800-1-12",..: 1..
 $ run      : int  1 5 2 2 4 3 5 2 1 3 ...
 $ gap      : num  3.3 3.3 5.5 5.5 3.3 ...
 $ class    : Factor w/ 6 levels "G-800-1-1","G-800-1-w",...
 $ inst.seed: Factor w/ 5 levels "1","2","3","4",..: 1 1 2 2..
```

Nesting is automatically handled appropriately in `lmer` as long as the levels of the instance factor are distinct (Bates 2007). This might be not the case when the nesting is implicit, that is, when the labels used for the levels of the variable at the inner state are incomplete. For example, this is the case if we identify the instances for each combination of the instance factors, type and weights, by the seeds used to generate them, say, `1`, `2`, `3`, `4`, `5`. We might have 5 seeds but $3 \times 2 \times 5$ different instances. If so, then we just need to specify the seed as the random factor by `(1|type:weights:seed)` or relabel the instances as `G-800-w-1`, `G-800-w-2`, etc. and specify simply `(1|type:weights)`. Our data have both identifiers described: `instance` and `inst.seed`.

We first test the hypothesis on the nested random effects. Again we can choose between F-ratio and likelihood ratio test. The likelihood ratio test has the advantage that it does not require to recalculate the expected mean squares for the appropriate test statistic, so we try that first. In R we have

```
> fm3.1 <- lmer(gap ~ (type + weights)^2 +
      (1 | type:weights:inst.seed), data = OPMR, REML = FALSE)
> fm3.0 <- lm(gap ~ (type + weights)^2 + 1, data = OPMR)
> LRT <- as.numeric(2 * (logLik(fm3.1) - logLik(fm3.0)))
> 1 - pchisq(LRT, 1)
```

```
[1] 1.71e-14
```

We see that we can reject $H_0 : \sigma_\tau^2 = 0$. As mentioned above, the likelihood ratio test is more conservative than the F-ratio test hence, since we reject already, there is no need to check the F-ratio test as well. We therefore include the random effect in the model.

The next step is considering the fixed factors that determine the instance classes. Again, since the experiment is balanced the p-values can be determined via the `anova` F statistics

```
> fm3 <- lmer(gap ~ (type + weights)^2 +
      (1 | type:weights:inst.seed), data = OPMR)
> fm3.aov <- anova(fm3)
> print(fm3.aov, digits = 3)
```

```
Analysis of Variance Table
             Df  Sum Sq Mean Sq F value
type          2  21.43   10.71   42.43
weights       1   0.10    0.10    0.38
type:weights  2   1.85    0.92    3.66
```

and manually derive the p-values adding the degrees of freedom of the denominator, that in this case are $(r-1)b_1b_2$, with b_1 and b_2 being the number of levels of the two instance factors

```
> type <- fm3.aov["type", ]
> 1 - pf(type$"F value", type$Df, (5 - 1) * 3 * 2)
```

```
[1] 1.32e-08
```

```
> weights <- fm3.aov["weights", ]
> 1 - pf(weights$"F value", weights$Df, (5 - 1) * 3 * 2)
```

```
[1] 0.544
```

```
> interaction <- fm3.aov["type:weights", ]
> 1 - pf(interaction$"F value", interaction$Df, (5 - 1) * 3 * 2)
```

```
[1] 0.0409
```

We conclude that the type has a significant effect on the average performance of the algorithm while the weights not. If relevant to the analysis, one can proceed to consider the estimated effects of these two fixed factors. They are to be interpreted as the estimated change in the mean lower bound approximation of the algorithm caused by different characteristics of the instances.

10.4.4 Case $\langle N, q(M), r\rangle$: General Design

In the last case, we aim at a general analysis of the influence on mean performance of local search components and different instance features. We consider a design with the following algorithm and instance factors

- `init.heur`: the construction heuristic with levels {`gc`,`la`,`sp`};
- `neigh`: the neighborhood with levels {`addn`,`gcn`,`spn`};
- `k`: the value of k in the neighborhoods with categorical levels {`1`,`3`,`5`};

- `type`: the type of graphs with levels {`U`,`G`,`sm`};
- `dens`: the edge density in the graph with levels {`l`,`m`,`h`};
- `weights`: the distribution of weights with levels {`w`,`1`}.

All these factors are fixed factors. Each combination of the three instance factors gives rise to a class from which we sample 5 instances. The additional factor `instance`, or `inst.seed`, is, therefore, a random factor. The experiment has $3 \times 3 \times 2 \times 5 = 90$ experimental units (instances). Moreover we replicate each run of an algorithm 5 times leading to a total of 12150 runs over all.

```
> load("Data/NPMR.dataR")
> str(NPMR, strict.width = "cut", width = 70)
```

```
'data.frame':        12150 obs. of  12 variables:
 $ weights  : Factor w/ 2 levels "1","w": 1 1 1 1 1 1 ...
 $ type     : Factor w/ 3 levels "G","U","sm": 1 1 1 1 1 ...
 $ dens     : Factor w/ 3 levels "h","l","m": 2 2 2 2 2 2 ...
 $ init.heur: Factor w/ 3 levels "gc","la","sp": 3 2 2 1 2..
 $ neigh    : Factor w/ 3 levels "addn","gcn","spn": 2 1 3 1..
 $ k        : Factor w/ 3 levels "1","3","5": 1 3 3 2 3 1 2..
 $ instance : Factor w/ 90 levels "G-800-h-1-1",..: 11 11 1..
 $ run      : int  4 4 5 2 3 2 3 3 3 3 ...
 $ gap      : num  11 3.3 3.3 5.5 6.6 ...
 $ class    : Factor w/ 18 levels "G-800-h-1","G-800-h-w",..: 3..
 $ algorithm: Factor w/ 27 levels "gc-addn-1","gc-addn-3",..:12..
 $ inst.seed: Factor w/ 5 levels "1","2","3","4",..: 1 1 1..
```

In a full nested factorial design of this kind we could study fixed effects interactions up to the sixth level. However, high order interactions are difficult to interpret and we, therefore, restrict ourselves to interactions of level three and to no interaction between algorithmic and instance factors.[7] Let's first test the significance of the instance factor.

```
> fm4.1 <- lmer(gap ~ (type + weights + dens)^3 + (init.heur +
     neigh + k)^3 + (1 | type:weights:dens:inst.seed),
     data = NPMR, REML = FALSE)
> fm4.0 <- lm(gap ~ (type + weights + dens)^3 + (init.heur +
     neigh + k)^3, data = NPMR)
> LRT <- as.numeric(2 * (logLik(fm4.1) - logLik(fm4.0)))
> 1 - pchisq(LRT, 1)
[1] 0.224
```

In this case the p-value from the likelihood ratio test is not significant and it does not allow us to reject the null hypothesis that the two models are equal. However, as we mentioned the likelihood ratio test is rather conservative, hence we check also the exact F-ratio test. The terms in the F-ratio are the same as those provided in Table 10.2 of Section 10.2.3 but the derivation of the degrees of freedom require some more work. Calling `a1`, ..., `aN` and `b1`, ..., `bM` the levels of the algorithmic factors and instance factors, respectively, and using the rules for the degrees of freedom given by Montgomery (2005, pag. 502) we obtain

```
> fm4.1 <- lmer(gap ~ (type + weights + dens)^3 + (init.heur +
     neigh + k)^3 + (1 | instance), data = NPMR)
> VC <- VarCorr(fm4.1)
> sigma.tau <- as.numeric(attr(VC$instance, "stddev"))
> sigma <- as.numeric(attr(VC, "sc"))
> F.ratio <- (sigma^2 + (a1 * a2 * a3 * r) * sigma.tau^2)/sigma^2
> (df1 <- b1 * b2 * b3 * (q - 1))
[1] 72
> (df2 <- as.numeric(fm4.1@dims["n"]) - 1 - sum(anova(fm4.1)["Df"])
     - df1)
[1] 12034
> 1 - pf(F.ratio, df1, df2)
[1] 0.00469
```

The p-value is significant. Since the test is exact we give preference to this result and proceed to analyze the fixed effects using the mixed model rather than the ordinary ANOVA. In order to add the p-values to the ANOVA table we use a function written by ourselves and available from the online compendium. This function is simply a wrapper that uses the F-ratio from the `anova` method for `lmer` and the degrees of freedom derived by the rules of Montgomery (2005).

[7] Note that if high order interactions are not of interest, fractional factorial designs are a better choice than full factorial designs because they minimize the number of experiments.

```
> fm4 <- lmer(gap ~ (type + weights + dens)^3 + (init.heur +
     neigh + k)^3 + (1 | type:weights:dens:inst.seed),
     data = NPMR)
> anova.4lmer.balanced(fm4, c("type", "weights", "dens"),
     instance.id = "inst.seed")
Analysis of Variance Table
                  Num. Def Sum Sq Mean Sq F value Den. Df Pr(>F)
type                     2  82058   41029  951.95      72 <2e-16
weights                  1  22441   22441  520.66      72 <2e-16
dens                     2   6882    3441   79.84      72 <2e-16
init.heur                2 142577   71288 1654.02   12034 <2e-16
neigh                    2 117939   58969 1368.19   12034 <2e-16
k                        2  76931   38465  892.47   12034 <2e-16
type:weights             2  90635   45318 1051.45      72 <2e-16
type:dens                4    313      78    1.82      72   0.14
weights:dens             2   9473    4736  109.89      72 <2e-16
init.heur:neigh          4  47798   11949  277.25   12034 <2e-16
init.heur:k              4  46994   11748  272.58   12034 <2e-16
neigh:k                  4  75897   18974  440.24   12034 <2e-16
type:weights:dens        4    209      52    1.21      72   0.31
init.heur:neigh:k        8  40766    5096  118.23   12034 <2e-16
```

The results indicate that all main effects are significant and that among the interactions only `type:weight:dens` and `type:dens` are not significant. The omission of effects that might be significant in the model may result in an overestimation of the denominator in the F-ratio and consequently in more conservative tests. However, these results are sufficient for us. They indicate that there is a significant effect of the nesting factors and this indicates that the analysis must be differentiated for each class.

Our final step is to split the data and to perform for each class an analysis similar to the one of case 2. In Figure 10.8 we report in a dotplot the average results of each algorithmic configuration on each instance class supported by confidence intervals (the function `intervals` to compute the overall plot is available online). The overall result is that the combination `gc-addn` is the one yielding the best performance, consitently over the different instance classes, while there does not seem to be a significant difference for this configuration in the value chosen for k.

10.5 Summary and Outlook

In this chapter, we described linear statistical models and their use in the specific task of analyzing the results of optimization algorithms. We put our emphasis on *mixed-effects models* in which algorithmic components are treated as fixed factors and the test instances as random factors. We provided evidence that these models lead to different inferences with respect to ordinary ANOVA models where the instances are treated also as fixed factors. In addition, we argued that when instance factors are also subject of study then the models become nested or, alternatively, separate analyses have to be conducted.

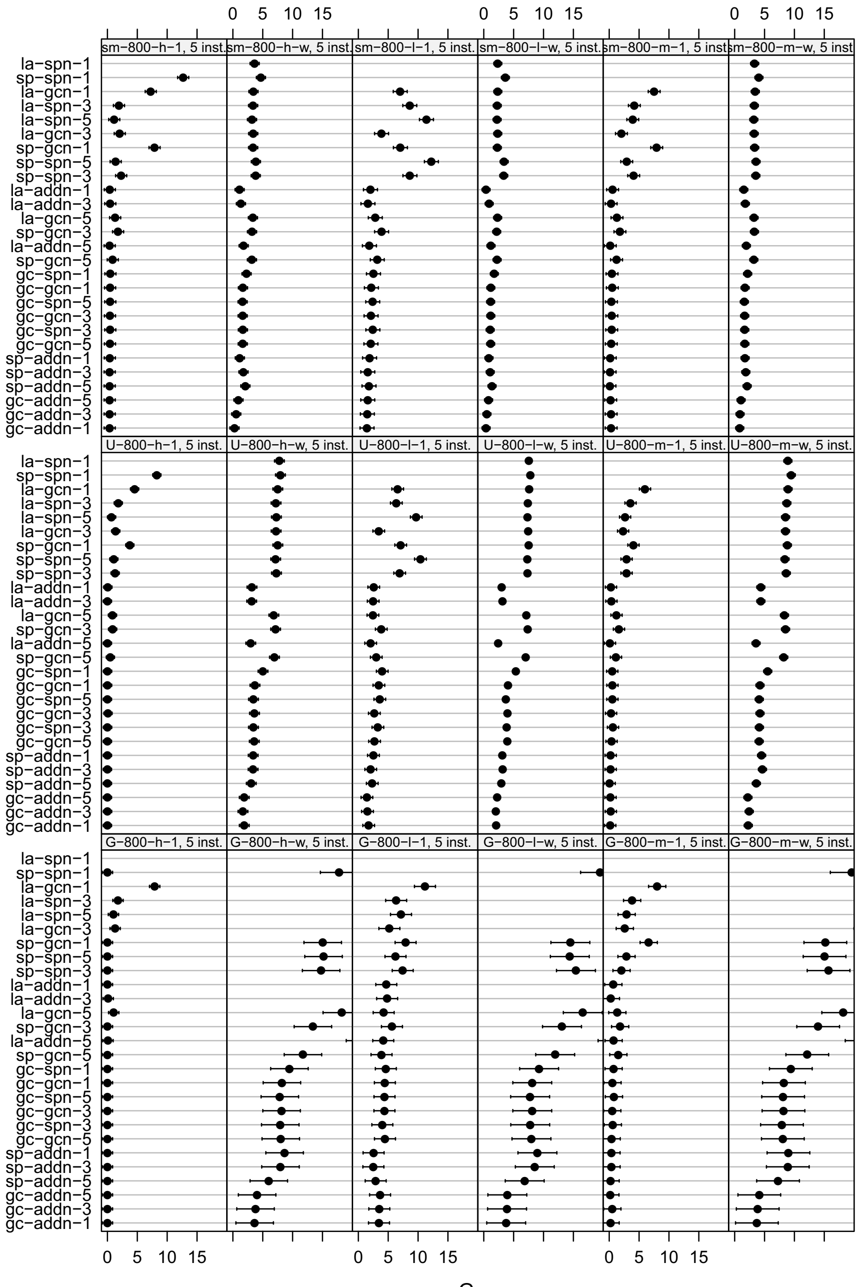


Fig. 10.8: Confidence intervals derived as described in Section 10.2.2 for each instance class indepenedently taken. On the y-axis the labels of the algorithms indicate the composition with respect to the components described in the text. On the x-axis, the gap represents the percentage deviation from the lower bound

We developed a detailed example for didactic purposes and showed how the results from the analysis of mixed models should be interpreted and how they may be presented by means of tables and graphics. These might be used in articles in addition to the common practice of reporting numerical results on a few benchmark instances. The inferential analysis becomes more relevant to be reported as the amount of data decreases.

There are a number of issues that we left out and that may be included in this framework. Examples are conditional parameters for the algorithms, which can be modeled similarly to the nesting of the instances, and the permutation of instance data, a feature that might have a certain impact on the results, which can also be included, yielding a design with two nested random factors and thus a hierarchical model (Fox 2002).

There are also further developments that could be pursued. The graphical presentation of results constitutes one of the possible improvements of this work. Regression trees offer a neat and concise way to attain this. They consist of (binary) trees obtained by branching the data under analysis, with branching higher in the tree for the factors responsible for the largest evidence for differences (identified by the entity of the p-value). However, all available packages of which we are aware do not include the possibility of treating nesting, and random factors or blocking factors.

This whole chapter was based on the assumption of additive *linear* models and *normality* of data. A natural extension of this work is the use of nonlinear mixed-effects models and generalized linear mixed-effects models that seem more appropriate in many cases of analysis of optimization algorithms. These models are often used in the study of repeated measurements over time of a certain response (longitudinal data). This could disclose a further development, that is, the analysis and comparison of optimization algorithms not only on the basis of their final response but also on the way performance changes over runtime.

References

Bang-Jensen J, Chiarandini M, Morling P (2009) A computational investigation of heuristic algorithms for 2-edge-connectivity augmentation. Networks (In print.)

Barr R, Golden B, Kelly J, Resende M, Stewart W (1995) Designing and reporting on computational experiments with heuristic methods. Journal of Heuristics 1(1):9–32

Bates D (2007) Personal Communication

Bates D, Maechler M, Dai B (2008) lme4: Linear mixed-effects models using S4 classes. URL `http://lme4.r-forge.r-project.org/`, R package version 0.999375-28

Birattari M (2004) On the estimation of the expected performance of a metaheuristic on a class of instances. How many instances, how many runs? Tech. Rep. TR/IRIDIA/2004-01, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium

Bondy J, Murty U (2008) Graph Theory. Graduate Texts in Mathematics, Vol. 244, Springer

Chiarandini M (2005) Stochastic local search methods for highly constrained combinatorial optimisation problems. PhD thesis, Computer Science Department, Darmstadt University of Technology, Darmstadt, Germany

Coffin M, Saltzman MJ (2000) Statistical analysis of computational tests of algorithms and heuristics. INFORMS Journal on Computing 12(1):24–44

Conforti M, Galluccio A, Proietti G (2004) Edge-connectivity augmentation and network matrices. In: Workshop on Graph-Theoretic Concepts in Computer Science, Springer, Lecture Notes in Computer Science, vol 3353, pp 355–364

Cormen T, Leiserson C, Rivest R (2001) Introduction to Algorithms, 2nd edn. MIT press

Czarn A, MacNish C, Vijayan K, Turlach B, Gupta R (2004) Statistical exploratory analysis of genetic algorithms. Evolutionary Computation, IEEE Transactions on 8(4):405–421

Fox J (2002) Linear mixed models. Appendix to An R and S-PLUS Companion to Applied Regression, URL `http://cran.r-project.org/doc/contrib/Fox-Companion/appendix-mixed-models.pdf`

Glover F, Kochenberger G (eds) (2002) Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol 57. Kluwer Academic, Norwell, MA

Hooker JN (1996) Testing heuristics: We have it all wrong. Journal of Heuristics 1:32–42

Johnson R, Wichern D (2007) Applied Multivariate Statistical Analysis, 6th edn. Prentice-Hall International

Kutner MH, Nachtsheim CJ, Neter J, Li W (2005) Applied Linear Statistical Models, 5th edn. McGraw-Hill

Lehmann E (2003) Theory of point estimation. Springer

Lehmann E, Romano J (2008) Testing statistical hypothesis. Springer

Lenth RV (2006) Java applets for power and sample size [computer software]. Retrieved 29 January 2009 from `http://www.stat.uiowa.edu/~rlenth/Power`

Lin BW, Rardin RL (1979) Controlled experimental design for statistical comparison of integer programming algorithms. Management Science 25(12):1258–1271

McGeoch CC (1996) Toward an experimental method for algorithm simulation. INFORMS Journal on Computing 8(1):1–15, this journal issue contains also commentaries by Pierre L'Ecuyer, James B. Orlin and Douglas R. Shier, and a rejoinder by C. McGeoch

Michiels W, Aarts E, Korst J (2007) Theoretical Aspects of Local Search. Monographs in Theoretical Computer Science, An EATCS Series, Springer

Molenberghs G, Verbeke G (eds) (1997) Linear Mixed Models in Practice - A SAS-Oriented Approach. Springer

Molenberghs G, Verbeke G (2005) Models for Discrete Longitudinal Data. Springer

Montgomery DC (2005) Design and Analysis of Experiments, 6th edn. Wiley

Pinheiro J, Bates D, DebRoy S, Sarkar D, the R Core team (2008) nlme: Linear and Nonlinear Mixed Effects Models. R package version 3.1-89

Pinheiro JC, Bates DM (2000) Mixed-Effects Models in S and S-Plus. Springer

R Development Core Team (2008) R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, URL `http://www.R-project.org`

Rardin RL, Uzsoy R (2001) Experimental evaluation of heuristic optimization algorithms: A tutorial. Journal of Heuristics 7(3):261–304

Rousseeuw PJ (1984) Least median of squares regression. Journal of the American Statistical Association 79(388):871–880, URL `http://www.jstor.org/stable/2288718`

SAS Institute Inc. (2007) SAS online documentation: Parameterization of mixed models. `http://www.webcitation.org/5h0u00trT`, retrieved on 2009-05-24

Stram D, Lee J (1994) Variance components testing in the longitudinal mixed effects model. Biometrics 50(4):1171–1177

Stram D, Lee J (1995) Correction to 'Variance components testing in the longitudinal mixed effects model'. Biometrics 51(3):1196

Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1(1):67–82

Zanakis SH (1977) Heuristic 0-1 linear programming: An experimental comparison of three methods. Management Science 24(1):91–104

Zemel E (1981) Measuring the quality of approximate solutions to zero-one programming problems. Mathematics of Operations Research 6(3):319–332

Chapter 11
Tuning an Algorithm Using Design of Experiments

Enda Ridge and Daniel Kudenko

Abstract This chapter is a tutorial on using a *design of experiments* approach for tuning the parameters that affect algorithm performance. A case study illustrates the application of the method and interpretation of its results.

11.1 Introduction

This chapter presents a case study of a methodology for selecting algorithm parameter values to tune algorithm performance. It efficiently takes the experimenter from the initial situation of almost no knowledge of the algorithm's behavior to the desired situation of an accurately modeled algorithm. This provides recommendations on tuning parameter settings for given problem characteristics. The methodology is based on well-established procedures from *design of experiments* (DOE) (Montgomery 2005) that have been modified for their application to algorithm tuning (Ridge 2007). The field of DOE is defined as:

> ...a systematic, rigorous approach to engineering problem-solving that applies principles and techniques at the data collection stage so as to ensure the generation of valid, defensible, and supportable engineering conclusions. In addition, all of this is carried out under the constraint of a minimal expenditure of engineering runs, time, and money. (Croarkin and Tobias 2006)

DOE is well established theoretically and well supported in terms of software tools. The traditional areas to which DOE is applied in engineering map almost directly to the common research questions that one asks in algorithm research so

Enda Ridge
Datalab, The Technology Innovation Group, Detica Ltd., London, UK
e-mail: Enda.Ridge@Detica.com

Daniel Kudenko
Department of Computer Science, The University of York, UK
e-mail: Kudenko@cs.york.ac.uk

DOE's power and maturity can be transferred directly to algorithm research. DOE offers efficiency in terms of the amount of data that needs to be gathered. This is critical when attempting to understand immense algorithm design spaces. All DOE conclusions are based on statistical analyses and so are supported with mathematical precision. This allays any concerns regarding subjective interpretation of results.

The case study applies DOE to an *ant colony system* (ACS) (Dorigo and Stützle 2004) for the *traveling salesperson problem* (TSP) (Lawler et al. 1995). Further details on the ACS algorithm are well covered in the literature. For our purposes here, it is sufficient to understand that ACS is a heuristic (randomized) optimization algorithm with many tuning parameters and at least two problem characteristics that affect its performance. Before reporting the case study, we first discuss some preliminaries that are important for an understanding of DOE.

11.2 Research Questions Addressed with DOE

Modeling algorithm performance is a sensible way to explore the vast design space of tuning parameter settings and their relationship to problem instances and algorithm performance. A good model can be used to quickly explore algorithm performance without resorting to expensive algorithm runs. Models permit addressing the following research questions:

- **Screening.** Which tuning parameters and which problem characteristics have no significant effect on the performance of the algorithm in terms of solution quality and solution time?
- **Ranking.** What is the relative importance of the most important tuning parameters and problem characteristics?
- **Relationship between tuning, problems, and performance.** What is the relationship between tuning parameters, problem characteristics, and the responses of solution quality and solution time? A tuning study yields a mathematical equation modeling this relationship for each response.
- **Tuned parameter settings.** What is a good set of tuning parameter settings given an instance with certain characteristics? Are these settings better than what can be achieved with randomly chosen settings? Are these settings better than alternative settings from the literature?

11.3 Experiment Designs

The design that an experimenter uses will depend on many things, including the particular research question, whether experiments are in the early stages of research, and the experimental resources available. This section focuses on the advanced designs that appear in this chapter. It begins with a simpler, more common design as this provides the necessary background for understanding the subsequent designs.

11.3.1 Full and 2^k Factorial Designs

A *full factorial* design consists of a crossing of all levels of all factors. A factor (or independent variable) is a variable that an experimenter varies. The number of levels of each factor can be two or more and need not be the same for each factor. The full factorial is an extremely powerful but expensive design. A more useful type of factorial for DOE uses k factors, each at only 2 levels. The so-called 2^k factorial design provides the smallest number of runs with which k factors can be studied in a full factorial design. Factorials have some particular advantages and disadvantages (Ostle 1963). These are worth noting given the importance that factorials play in DOE experimental design. The advantages are that:

- Greater efficiency is achieved in the use of available experimental resources in comparison with what could be learned from the same number of experiment runs in a less structured context such as a *one-factor-at-a-time* analysis,
- Information is obtained about the interactions, if any, of factors because the factor levels are all crossed with one another. An interaction is where the experimental response for a given factor level cannot be understood without also specifying the level of an interacting factor.

Of course, these advantages come at a price. As the number of factors grows, the number of combinations of factor levels (treatments) in a 2^k design rapidly overwhelms the experiment resources. Consider the case of 10 continuous factors. A naïve full factorial design for these ten factors will require a prohibitive $2^{10} = 1024$ treatments. A more efficient design is required.

11.3.2 Fractional Factorial Designs

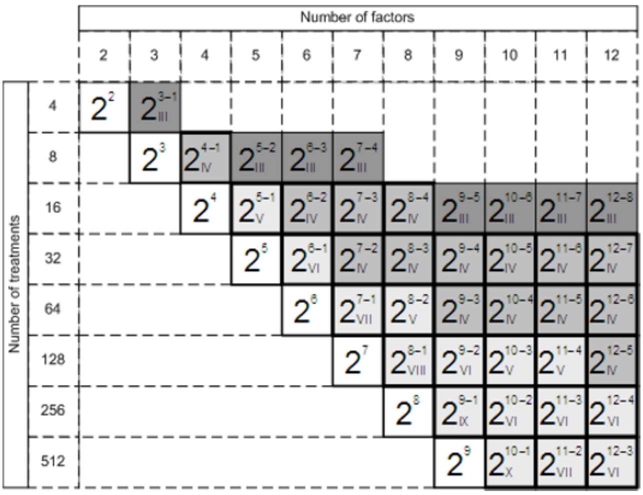
There are benefits to the expense of a full factorial design. A 2^{10} full factorial will provide data to evaluate all the effects listed in Table 11.1.

If it is assumed that higher-order interactions are insignificant, information on the main effects and lower-order interactions can be obtained by running a fraction of the complete factorial design. This assumption is based on the *sparsity of effects principle* (Wu and Hamada 2000). This states that a system or process is likely to be most influenced by some main effects and low-order interactions and less influenced by higher-order interactions.

A judiciously chosen fraction of the treatments in a full factorial will yield insights into only the lower-order effects. This is termed a *fractional factorial*. The price we pay for the fractional factorial's reduction in number of experimental treatments is that some effects are indistinguishable from one another; they are *aliased*. Additional treatments, if necessary, can disentangle these aliased effects should an alias group be statistically significant. The advantage of the fractional factorial is that it facilitates sequential experimentation. The additional treatments and associated experiment runs need only be performed if aliased effects are statistically

Effect	**Number of effects estimated**
Main	10
Two-factor	45
Three-factor	120
Four-factor	210
Five-factor	252
Six-factor	210
Seven-factor	120
Eight-factor	45
Nine-factor	10
Ten-factor	1

Table 11.1: Number of each effect estimated by a full factorial design of 10 factors

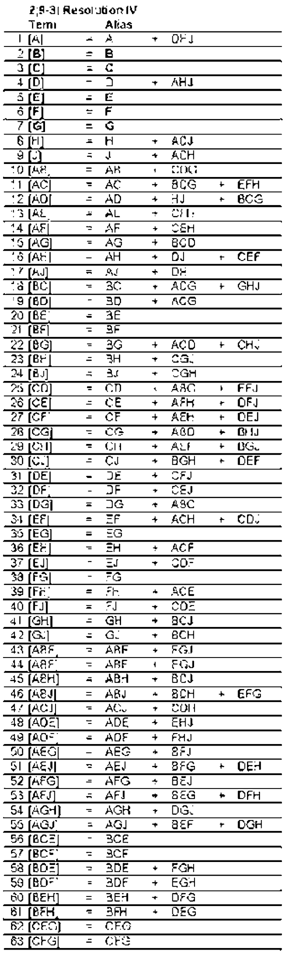


Number of treatments	Number of factors										
	2	3	4	5	6	7	8	9	10	11	12
4	2^2	2^{3-1}_{III}									
8		2^3	2^{4-1}_{IV}	2^{5-2}_{III}	2^{6-3}_{III}	2^{7-4}_{III}					
16			2^4	2^{5-1}_{V}	2^{6-2}_{IV}	2^{7-3}_{IV}	2^{8-4}_{IV}	2^{9-5}_{III}	2^{10-6}_{III}	2^{11-7}_{III}	2^{12-8}_{III}
32				2^5	2^{6-1}_{VI}	2^{7-2}_{IV}	2^{8-3}_{IV}	2^{9-4}_{IV}	2^{10-5}_{IV}	2^{11-6}_{IV}	2^{12-7}_{IV}
64					2^6	2^{7-1}_{VII}	2^{8-2}_{V}	2^{9-3}_{IV}	2^{10-4}_{IV}	2^{11-5}_{IV}	2^{12-6}_{IV}
128						2^7	2^{8-1}_{VIII}	2^{9-2}_{VI}	2^{10-3}_{V}	2^{11-4}_{V}	2^{12-5}_{IV}
256							2^8	2^{9-1}_{IX}	2^{10-2}_{VI}	2^{11-3}_{VI}	2^{12-4}_{VI}
512								2^9	2^{10-1}_{X}	2^{11-2}_{VII}	2^{12-3}_{VI}

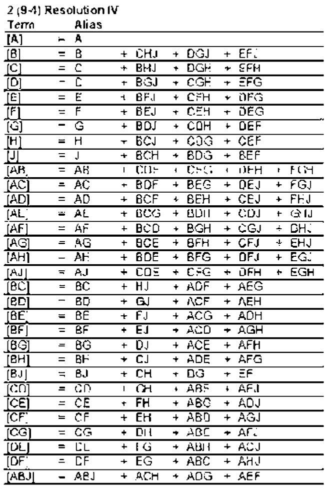
Fig. 11.1: Fractional factorial designs for 2 to 12 factors. The required number of treatments is listed on the left. Resolution III designs (do not estimate any terms) are colored darkest followed by Resolution IV designs (estimate main effects only), followed by Resolution V and higher (estimate main effects and second-order interactions)

significant. Depending on the number of factors, and consequently the design size, a range of fractional factorials can be produced from a full factorial.

The amount of higher-order effects that are aliased is described by the design's *resolution*. For Resolution III designs, all effects are aliased. Resolution IV designs have unaliased main effects but second-order effects are aliased. Resolution V designs estimate main and second-order effects without aliases. The details of how to choose a fractional factorial's treatments are beyond the scope of this chapter. It is an established algorithmic procedure that is well covered in the literature (Montgomery 2005) and is provided in all modern statistical analysis software. The fractional factorials used in this case study are summarized in Fig. 11.1 which shows the


2(9-3) Resolution IV

Term		Alias				
1 [A]	=	A	+	DFJ		
2 [B]	=	B				
3 [C]	=	C				
4 [D]	=	D	+	AHJ		
5 [E]	=	E				
6 [F]	=	F				
7 [G]	=	G				
8 [H]	=	H	+	ACJ		
9 [J]	=	J	+	ACH		
10 [AB]	=	AB	+	CDG		
11 [AC]	=	AC	+	BCG	+	EFH
12 [AD]	=	AD	+	HJ	+	BCG
13 [AE]	=	AE	+	CFH		
14 [AF]	=	AF	+	CEH		
15 [AG]	=	AG	+	BCD		
16 [AH]	=	AH	+	DJ	+	CEF
17 [AJ]	=	AJ	+	DH		
18 [BC]	=	BC	+	ACG	+	GHJ
19 [BD]	=	BD	+	ACG		
20 [BE]	=	BE				
21 [BF]	=	BF				
22 [BG]	=	BG	+	ACD	+	CHJ
23 [BH]	=	BH	+	CGJ		
24 [BJ]	=	BJ	+	CGH		
25 [CD]	=	CD	+	ABG	+	EFJ
26 [CE]	=	CE	+	AFH	+	DFJ
27 [CF]	=	CF	+	AEH	+	DEJ
28 [CG]	=	CG	+	ABD	+	BHJ
29 [CH]	=	CH	+	AEF	+	BGJ
30 [CJ]	=	CJ	+	BGH	+	DEF
31 [DE]	=	DE	+	CFJ		
32 [DF]	=	DF	+	CEJ		
33 [DG]	=	DG	+	ABC		
34 [EF]	=	EF	+	ACH	+	CDJ
35 [EG]	=	EG				
36 [EH]	=	EH	+	ACF		
37 [EJ]	=	EJ	+	CDF		
38 [FG]	=	FG				
39 [FH]	=	FH	+	ACE		
40 [FJ]	=	FJ	+	CDE		
41 [GH]	=	GH	+	BCJ		
42 [GJ]	=	GJ	+	BCH		
43 [ABF]	=	ABF	+	FGJ		
44 [ABF]	=	ABF	+	FGJ		
45 [ABH]	=	ABH	+	BCJ		
46 [ABJ]	=	ABJ	+	BCH	+	EFG
47 [ACJ]	=	ACJ	+	CDH		
48 [ADE]	=	ADE	+	EHJ		
49 [ADF]	=	ADF	+	FHJ		
50 [AEG]	=	AEG	+	BFJ		
51 [AEJ]	=	AEJ	+	BFG	+	DEH
52 [AFG]	=	AFG	+	BEJ		
53 [AFJ]	=	AFJ	+	BEG	+	DFH
54 [AGH]	=	AGH	+	DGJ		
55 [AGJ]	=	AGJ	+	BEF	+	DGH
56 [BCE]	=	BCE				
57 [BCF]	=	BCF				
58 [BDE]	=	BDE	+	FGH		
59 [BDF]	=	BDF	+	EGH		
60 [BEH]	=	BEH	+	DFG		
61 [BFH]	=	BFH	+	DEG		
62 [CEG]	=	CEG				
63 [CFG]	=	CFG				


2 (9-4) Resolution IV

Term		Alias								
[A]	=	A								
[B]	=	B	+	CHJ	+	DGJ	+	EFJ		
[C]	=	C	+	BHJ	+	DGH	+	EFH		
[D]	=	D	+	BGJ	+	CGH	+	EFG		
[E]	=	E	+	BFJ	+	CFH	+	DFG		
[F]	=	F	+	BEJ	+	CEH	+	DEG		
[G]	=	G	+	BDJ	+	CDH	+	DEF		
[H]	=	H	+	BCJ	+	CDG	+	CEF		
[J]	=	J	+	BCH	+	BDG	+	BEF		
[AB]	=	AB	+	CDF	+	CEG	+	DEH	+	FGH
[AC]	=	AC	+	BDF	+	BEG	+	DEJ	+	FGJ
[AD]	=	AD	+	BCF	+	BEH	+	CEJ	+	FHJ
[AE]	=	AE	+	BCG	+	BDH	+	CDJ	+	GHJ
[AF]	=	AF	+	BCD	+	BGH	+	CGJ	+	DHJ
[AG]	=	AG	+	BCE	+	BFH	+	CFJ	+	EHJ
[AH]	=	AH	+	BDE	+	BFG	+	DFJ	+	EGJ
[AJ]	=	AJ	+	CDE	+	CFG	+	DFH	+	EGH
[BC]	=	BC	+	HJ	+	ADF	+	AEG		
[BD]	=	BD	+	GJ	+	ACF	+	AEH		
[BE]	=	BE	+	FJ	+	ACG	+	ADH		
[BF]	=	BF	+	EJ	+	ACD	+	AGH		
[BG]	=	BG	+	DJ	+	ACE	+	AFH		
[BH]	=	BH	+	CJ	+	ADE	+	AFG		
[BJ]	=	BJ	+	CH	+	DG	+	EF		
[CD]	=	CD	+	GH	+	ABF	+	AFJ		
[CE]	=	CE	+	FH	+	ABG	+	ADJ		
[CF]	=	CF	+	EH	+	ABD	+	AGJ		
[CG]	=	CG	+	DH	+	ABE	+	AFJ		
[DE]	=	DE	+	FG	+	ABH	+	ACJ		
[DF]	=	DF	+	EG	+	ABC	+	AHJ		
[ABJ]	=	ABJ	+	ACH	+	ADG	+	AEF		

Fig. 11.2: Effects and alias chains for a 2(9-3) resolution IV design and a 2(9-4) resolution IV design

relationship between number of factors, design resolution, and associated number of experiment treatments.

A resolution V design is preferable when resources allow because it tells us what second-order effects are present without the need for additional treatments and experiment runs. It is informative to consider the two available resolution IV designs for 9 factors in Fig. 11.2 as examples of the importance of examining alias structure.

The 2(9-4) design requires 32 treatments while the 2(9-3) is more expensive with 64 treatments. The cheaper 2(9-4) design has 8 of its 9 main effects aliased with 3 third-order interactions. The 2(9-3) design has only 4 of its 9 main effects aliased with a single third-order interaction. The second-order interactions are almost all

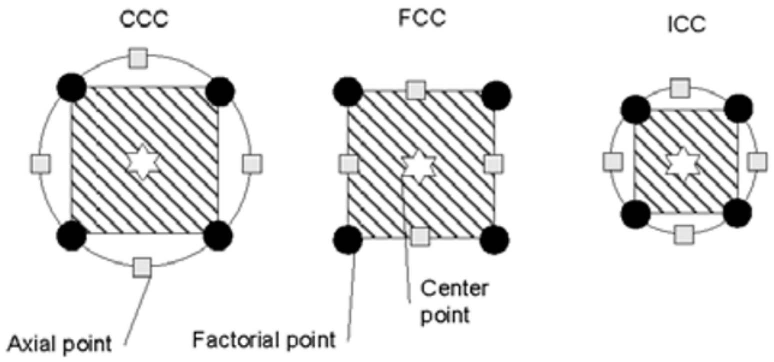


Fig. 11.3: Central composite designs for building response surface models. From left to right these designs are the circumscribed central composite (CCC), the face-centred composite (FCC) and the inscribed central composite (ICC). The design space is represented by the shaded area. The factorial points are black circles and the star points are grey squares

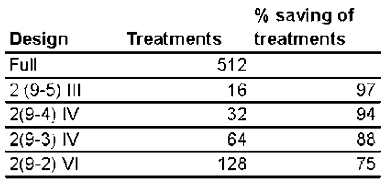
aliased in the more expensive 2(9-3) design but the aliasing is more favorable than in the cheaper 2(9-4) design.

11.3.3 Response Surface Designs

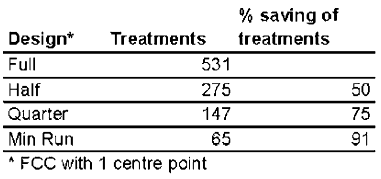
There are several types of experiment design for building response surface models. This chapter's case study uses *central composite designs* (CCD). A CCD contains an imbedded factorial (or fractional factorial design). This is augmented with both center points and a group of so-called *star points* (or axial points) that allow estimation of curvature in the resulting model. There are three types of central composite design, illustrated in Fig. 11.3.

The choice of design depends on the nature of the factors to study:

- **Circumscribed central composite (CCC).** In this design, the star points establish new extremes for the low and high settings for all factors. These designs require 5 levels for each factor. Augmenting an existing factorial or resolution V fractional factorial design with star points can produce this design.
- **Inscribed central composite (ICC).** For those situations in which the limits specified for factor settings are truly limits, the ICC design uses the factor settings as the star points and creates a factorial or fractional factorial design within those limits. This design also requires 5 levels of each factor.



Design	Treatments	% saving of treatments
Full	512	
2 (9-5) III	16	97
2(9-4) IV	32	94
2(9-3) IV	64	88
2(9-2) VI	128	75



Design*	Treatments	% saving of treatments
Full	531	
Half	275	50
Quarter	147	75
Min Run	65	91

* FCC with 1 centre point

Table 11.2: Savings in experiment runs. The savings for screening designs are on the left and the savings for response surface designs are on the right. In both cases, fractional factorial designs offer enormous savings in number of treatments over the full factorial alternative. "Half" and "quarter" refer to the fraction of the full design used. "Min Run" is a further extension to this concept permitting even greater run savings

- **Face-centred Composite (FCC).** In this design, the star points are at the center of each face of the factorial space. This design requires just 3 levels of each factor.

An existing factorial or resolution V design from the screening stage can be augmented with appropriate star points to produce the CCC and FCC designs. This is not the case with the ICC and so it is less useful in a sequential experimentation scenario.

11.3.4 Efficiency of Fractional Factorial Designs

Table 11.2 makes explicit the huge savings in experiment runs when using a fractional factorial design instead of a full factorial design.

11.4 Error, Significance, Power, and Replicates

Two types of error can be committed when testing hypotheses (Montgomery 2005, p. 35). If the null hypothesis is rejected when it is actually true, then a *type I error* has occurred. If the null hypothesis is not rejected when it is false then a *type II error* has occurred. These error probabilities are given special symbols:

- $\alpha = \Pr\{\text{Type I error}\} = \Pr\{\text{reject } H_0 | H_0 \text{ true}\}$
- $\beta = \Pr\{\text{Type II error}\} = \Pr\{\text{fail to reject } H_0 | H_0 \text{ false}\}$

In the context of type II errors, it is more convenient to use the *power* of a test, where

$$\text{Power} = 1 - \beta = \Pr\{\text{reject } H_0 |\ H_0 \text{ false}\}\,.$$

It is therefore desirable to have a test with a low α and a high power. The probability of a Type I Error is often called the *significance level* of a test. The particular significance level depends on the requirements of the experimenter and, in a research context, on the conventional acceptable level. Unfortunately, with so little adaptation of statistical methods to the analysis of heuristics, there are few guidelines on what value to choose. The power of a test is usually set to 80% by convention. The reason for this choice is due to diminishing returns. It requires an exponentially increasing number of replicates to increase power beyond about 80% and there is little advantage to the additional power this confers.

Miles[1] describes the relationship between significance level, effect size, sample size, and power using an analogy with searching.

- **Significance level:** This is the probability of thinking we have found something when it is not really there. It is a measure of how willing we are to risk a type I error.
- **Effect size:** The size of the effect in the population. The bigger it is, the easier it will be to find. This is a measure of the practical significance of a result, preventing us claiming a statistically significant result that has little consequence (Rardin and Uzsoy 2001).
- **Sample size:** A larger sample size leads to a greater ability to find what we were looking for. The harder we look, the more likely we are to find it.

The critical point regarding this relationship is that what we are looking for is always going to be there—it might just be there in such small quantities that we are not bothered about finding it. Conversely, if we look hard enough, we are guaranteed to find what we are looking for. Power analysis allows us to make sure that we have looked reasonably hard enough to find it. A typical experiment design approach is to agree the significance level and choose an effect size based on practical experience and experiment goals. Given these constraints, the sample size is increased until sufficient power is reached. If a response has a high variability then a larger sample size will be required.

Different statistical tests and different experiment designs involve different power calculations. These calculations can become quite involved and the details of their calculation are beyond the scope of this chapter. Power calculations are supplied with most good-quality statistical analysis software.

11.5 Benchmarking the Experimental Testbed

Clearly, all machines for computational experiments can differ widely. There are differences in processor speeds, memory sizes, chip types, operating systems, operating system versions, and in the case of Java different versions of different virtual

[1] "Getting the sample size right: a brief introduction to power analysis", http://www.jeremymiles.co.uk/misc/power

			Time (s)						
Instance	Size	Repetitions	116	253	111	156	188	136	96
E1k.0	1000	1000	5.45	4.38	5.25	5.00	3.31	4.81	3.37
E3k.0	3000	316	5.67	4.61	7.61	5.25	3.78	5.23	3.75
E10k.0	10000	100	6.91	7.25	8.81	6.44	4.99	6.64	5.32
E31k.0	31000	32	11.77	16.52	13.41	11.20	9.48	11.00	10.84
E100k.0	100000	10	22.86	26.53	26.77	21.03	10.87	19.85	12.82
E316k.0	316000	3	28.61	32.05	34.50	27.61	12.56	25.86	14.70
E1M.0	1000000	1	39.52	44.80	49.23	38.03	16.55	35.44	19.31

Table 11.3: Data from the DIMACS benchmarking of the experiment testbed. Running times are presented for the 7 experimental machines with IDs 116, 253, 111, 156, 188, 136, and 96

machines. Even if machines are identical in terms of all of these aspects, they may still differ in terms of running background processes such as virus checkers. This is the unfortunate reality of the majority of computational research environments. Furthermore, such differences will almost certainly occur in the computational resources of other researchers who attempt to reproduce or extend previous work of others. These differences necessitate the benchmarking of the experimental testbed.

Reproducibility of results is the motivation for benchmarking. Other researchers can reproduce the benchmarking process on their own experimental machines. They can thus better interpret the CPU times reported in this research by scaling them in relation to their own benchmarking results. This mitigates the decline in relevance of reported CPU times with inevitable improvements in technology.

The clear and simple benchmarking procedure of the DIMACS (Goldwasser et al. 2002) challenge was applied for this chapter's case study. The results are presented in Table 11.3. If other researchers reproduce the DIMACS procedure on their own machines then their numbers can be compared with Table 11.3 to scale the results.

11.6 Case Study

This section reports the chapter's case study on tuning the ACS algorithm with DOE. It is a template for how such DOE experiments could be reported since standardized reporting in other fields has greatly helped the interpretation of research results.

11.6.1 Problem Instances

All TSP instances were of the Euclidean symmetric type. In the Euclidean TSP, cities are points with integer coordinates in the two-dimensional plane. A cost matrix defines the distances between all cities in the problem instance. The TSP problem instances ranged in size from 300 cities to 500 cities with cost matrix standard deviation ranging from 10 to 70. All instances had a cost matrix mean of 100. The same instances were used for each replicate of a design point. Instances were generated with a version of the publicly available portmgen problem generator from the DIMACS challenge (Goldwasser et al. 2002).

11.6.2 Stopping Criterion

The choice of how to halt an experiment affects the results of an algorithm and thus the conclusions that can be drawn from the experiments. In this case study, experiments were halted after a stagnation stopping criterion. Stagnation was defined as a fixed number of iterations in which no improvement in solution value had been obtained. Responses were measured at several levels of stagnation during an experiment run: 50, 100, 150, 200, and 250 iterations. This facilitated examining the data at alternative stagnation levels to ensure that conclusions were the same regardless of stagnation level. An examination of the descriptive statistics verifies that the stagnation level did not have a large effect on the response values and therefore the conclusions after a 250 iteration stagnation should be the same as after lower iteration stagnations.

11.6.3 Response Variables

For experiments with algorithms, the response variables usually reflect some measure of solution quality and some measure of solution time. Two response variables were measured. The time in seconds to the end of an experiment reflects the *solution time*. The *relative error* from a known optimum reflects the solution quality. Concorde (Applegate et al. 2003) was used to calculate the optima of the generated instances. One may wonder why one would use a heuristic on a problem where the optimal solution can be calculated. The intention here is to evaluate a design of experiments methodology in a controlled manner.

11.6.4 Factors, Levels and Ranges

11.6.4.1 Held-Constant Factors

There are several held-constant factors. Local search, a technique typically used in combination with ACS, was omitted. All instances had a cost matrix mean of 100. The computation_limit parameter (Dorigo and Stützle 2004) was fixed at being limited to the candidate list length as this resulted in significantly lower solution times.

Factor	Name	Type	Low level	High level
A	Alpha	Numeric	1	13
B	Beta	Numeric	1	13
C	antsFraction	Numeric	1.00	110.00
D	nnFraction	Numeric	2.00	20.00
E	q0	Numeric	0.01	0.99
F	Rho	Numeric	0.01	0.99
G	rhoLocal	Numeric	0.01	0.99
H	solutionConstruction	Categoric	parallel	sequential
J	antPlacement	Categoric	random	same
K	pheromoneUpdate	Categoric	bestSoFar	bestOfIteration
L	problemSize	Numeric	300	500
M	problemStDev	Numeric	10.00	70.00

Table 11.4: Design factors for the tuning case study with ACS. The factor ranges are also given

11.6.4.2 Nuisance Factors

A limitation on the available computational resources necessitated running experiments across a variety of machines with slightly different specifications. Runs were executed in a randomized order across these machines to counteract any uncontrollable nuisance factors due to the background processes and differences in machine specification.

11.6.4.3 Design Factors

The design factors are the algorithm tuning parameters and problem characteristics whose relationship to performance metrics will be modeled by the DOE response surfaces. This case study examined 12 design factors. These factors and their high and low levels are listed in Table 11.4. Note that the two problem instance characteristics are included in the experiment design as we are modeling the relationship between the tuning parameters and various problems.

These parameters are discussed in further detail in the literature (Ridge 2007, Dorigo and Stützle 2004). For this case study, we are interested in the tuning parameters primarily as experiment design factors.

11.6.4.4 Experiment Design, Significance, Power and Replicates

The experiment design was a minimum run resolution V face-centred composite with six center points. A significance (alpha) level of 5% is used in this case study.[2]

[2] The value 5% is not a universally recommended significance level. The choice of alpha will depend on the statistical confidence required to the results. As discussed earlier, the cost of increased confidence is an increased number of experiment replicates.

Iterations		Time	Relative Error
50	Mean	65.33	11.01
	StDev	194.96	19.49
	Max	3131.77	125.84
	Min	0.17	0.55
	Actual Effect Size of 0.2 stdevs	**38.99**	**3.90**
100	Mean	136.38	10.73
	StDev	469.81	19.09
	Max	7825.44	124.20
	Min	0.28	0.55
	Actual Effect Size of 0.2 stdevs	**93.96**	**3.82**
150	Mean	204.39	10.57
	StDev	681.54	18.95
	Max	12075.97	124.20
	Min	0.38	0.47
	Actual Effect Size of 0.2 stdevs	**136.31**	**3.79**
200	Mean	270.25	10.47
	StDev	906.16	18.85
	Max	15423.77	124.20
	Min	0.50	0.46
	Actual Effect Size of 0.2 stdevs	**181.23**	**3.77**
250	Mean	341.36	10.40
	StDev	1121.20	18.81
	Max	15573.66	123.74
	Min	0.61	0.46
	Actual Effect Size of 0.2 stdevs	**224.24**	**3.76**

Table 11.5: Descriptive statistics for the ACS FCC design. The actual detectable effect size of 0.2 standard deviations is shown for each response and for each stagnation point

The number of replicates in the design is increased and further data are gathered until a power of 80% is reached. The effect size detectable at this combination of significance and power is then examined. Replicates were introduced into the design until an appropriate effect size was detectable. This approach of increasing replicates is known as a *work-up* procedure (Czarn et al. 2004). Note that it is more common to fix effect size and significance, increasing replicates until sufficient power is reached. The principle nonetheless remains the same.

The size of effect that could feasibly be detected depended on the particular response and the particular experiment design. Table 11.5 gives the descriptive statistics for the collected data and the actual effect size for each response. The design could achieve sufficient power with 5 replicates while detecting an effect of size 0.2 standard deviations in the response value.

At this stage, we have sufficient data to build a response surface model of *each individual* response. This model will be able to detect effects of the given effect size with a given confidence and power. We will ultimately be combining both the response models' recommendations into a simultaneous tuning of all the responses.

11.6.5 Model Fitting

The highest-order response surface model that can be generated from the FCC design used in this case study is quadratic. All lower-order models (linear and 2-factor interaction) are generated. If the model is not significant, it is removed from consideration and the next highest order model is examined for significance.

1. **Find important effects.** A stepwise linear regression is performed on the chosen model to estimate its coefficients. The stepwise regression identifies the model terms that can safely be removed, giving the most *parsimonious* model possible.
2. **Diagnosis.** The usual diagnostics of the proposed linear regression model are performed. This ensures that the model assumptions have not been violated.
3. **Normality.** A normal plot of studentized residuals should be approximately a straight line. Deviations from this may indicate that a transformation of the response is appropriate.
4. **Constant variance.** A plot of Studentized residuals against predicted response values should be a random scatter. Patterns such as a "megaphone" may indicate the need for a transformation of the response.
5. **Time-dependent effects.** A plot of Studentised residuals against run order should be a random scatter. Any trend indicates the influence of some time-dependent nuisance factor that was not countered with randomization.
6. **Model fit.** A plot of predicted values against actual response values will identify particular treatment combinations that are not well predicted by the model. Points should align along the 45° axis.
7. **Leverage and influence.** Leverage measures the influence of an individual design point on the overall model. A plot of leverage for each treatment indicates any problem data points.
8. A plot of **Cook's distance** against treatment measures how much the regression changes if a given case is removed from the model.

If the model passes these tests then its proposed coefficients can be accepted. If the model does not pass its diagnostics, there are two main options:

1. **Response transformation.** A transformation of the response may be required. Transformation simply means taking some function of the response variable such as the log or square root. The appropriate transformation can be identified using a *Box-Cox plot*.
2. **Outliers.** If the transformed response is still failing the diagnostics, it may be that there are outliers in the data. These should be identified and removed from the data. In this case study, outliers were deleted and the model building repeated until the models passed the diagnostics. Of the total data, 122 data points ($\sim$5%) were removed when analyzing the models of *relative error* and *time*.

It is always good practice to independently confirm the models' accuracy on some real data, different from the data used to generate the model.

As in traditional DOE, *confirmation* is achieved by running experiments at new randomly chosen points in the design space and comparing the actual data with the model's predictions. Confirmation is not a new rigorous experiment and analysis in itself but rather a quick informal check. In the case of an algorithm, these randomly chosen points in the design space equate to new problem instances and new randomly chosen combinations of tuning parameters. The methodology is as follows:

1. **Treatments.** A number of treatments are chosen where a treatment consists of a new problem instance and a new set of randomly chosen tuning parameter values with which the instance will be solved.
2. **Generate instances.** The required problem instances are generated.
3. **Random run order.** A random run order is generated for the treatments and a given number of replicates. Three replicates are often enough to give an estimate of how variable the response is for a given treatment. We are conducting this confirmation to ensure that our subjective decisions in the model building were correct.
4. **Prediction intervals.** The collected data for each response is compared with their respective models' 95% high and low prediction intervals (Montgomery 2005, p. 394-396).[3] Two criteria upon which our satisfaction with the models (and thus confidence in their predictions) can be judged are (Ridge and Kudenko 2007):

    - **Conservative:** we should prefer models that provide consistently higher predictions of relative error and longer solution time than those actually observed. We typically wish to minimize these responses and so a conservative model will predict these responses to be higher than their true value.
    - **Matching trend:** we should prefer models that match the trends in heuristic performance. The model's predictions of the parameter combinations that give the best and worst performance should match the combinations that yield the actual algorithm's observed best and worst performance.

5. **Confirmation.** If the models are not a satisfactory predictor of the actual algorithm then the experimenter must return to the model-building phase and attempt to improve the model.

The randomly chosen treatments produced actual algorithm responses with the descriptives listed in Table 11.6.

The large ranges of each response reinforce the motivation for correct parameter tuning as there is clearly a high cost in incorrectly tuned parameters. Figure 11.4 illustrates the 95% prediction intervals and actual confirmation data for the response surface models of *relative error* and *time*.

Looking at the predictions in general we see that time was sometimes better predicted than was the relative error. The *solution time* model matches all the trends

[3] The model's $p\%$ prediction interval is the range in which you can expect any individual value from the actual algorithm to fall into $p\%$ of the time. The prediction interval will be larger (a wider spread) than a confidence interval about averages since there is more scatter in individual values than in averages.

Iterations		Time	Relative Error
100	Mean	70.14	7.01
	StDev	84.14	3.48
	Max	528.28	17.65
	Min	1.55	3.12
150	Mean	109.80	6.88
	StDev	130.37	3.41
	Max	774.39	17.65
	Min	2.17	2.77
200	Mean	169.66	6.75
	StDev	220.76	3.38
	Max	1084.58	16.92
	Min	2.83	2.77
250	Mean	220.19	6.69
	StDev	287.57	3.35
	Max	1652.54	16.84
	Min	3.45	2.77

Table 11.6: Descriptive statistics for the confirmation of the ACS tuning. The response data is from runs of the actual algorithm on the randomly generated confirmation treatments

in the actual data. The *relative error* model however exhibits some false peaks and misses some actual peaks.

11.6.6 Results

11.6.6.1 Screening and Relative Importance of Factors

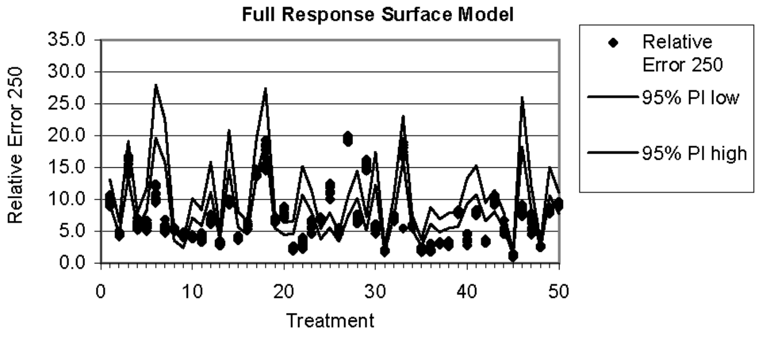
Figures 11.5 and 11.6 give the ranked ANOVAs of the *relative error* and *time* models from the analysis. The terms have been rearranged in order of decreasing sum of squares so that the largest contributor to the models comes first.

Looking first at the *relative error* rankings of Fig. 11.5, we see that the least important main effects are L-antPlacement, A-alpha, F-rho, and M-pheromoneUpdate.

By far the most important terms are the main effects of the exploration/exploitation tuning parameter (E) and the candidate list length tuning parameter (D) as well as their interaction. This is a very important result because it shows that candidate list length, a parameter that we have often seen set at a fixed value or not used, is actually one of the most important parameters to set correctly.

Looking at the *time* rankings of Fig. 11.6, we see that L-antPlacement was completely removed from the model. The least important main effects were then F-rho and A-alpha.

By far the most important tuning parameters are the number of ants and the lengths of their candidate lists. This is quite intuitive as the number of processing is directly related to these parameters. The result regarding the cost of the amount of ants is particularly important because the number of ants does not have a relatively



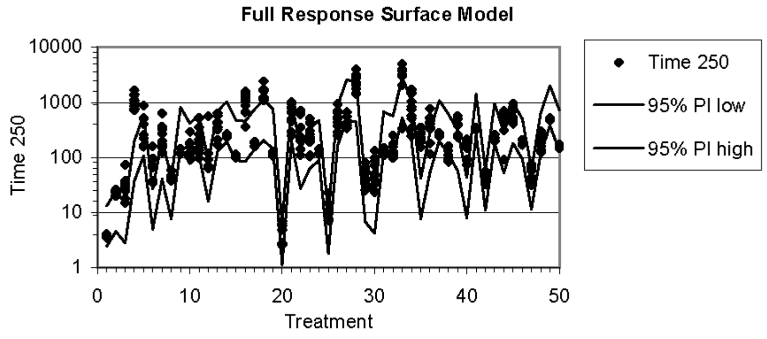


Fig. 11.4: The 95% prediction intervals for the ACS response surface model of relative error and time. The horizontal axis is the randomly generated treatment. The vertical axis is the relative error or time response

strong effect on solution quality. The extra time cost of using more ants will not result in gains in solution quality. This is an important result because it methodically confirms the often recommended parameter setting of setting the number of ants equal to a small number (usually 10).

11.6.6.2 Tuning

Since the response surface models are mathematical functions of the tuning parameters, it is possible to numerically optimize the models' responses by varying the tuning parameters. This allows us to produce the most efficient process. There are several possible optimization goals. We may wish to achieve a response with a given value (target value, maximum or minimum). Alternatively, we may wish that the response always falls within a given range (*relative error* less than 10%). More usually, we may wish to optimize several responses because of the algorithm compromise of quality and time. In the literature, tuning rarely deals with both so-

Rank	Term	Sum of squares	F value	p value
1	J-problemStDev	182.23	121362.13	< 0.0001
2	E-q0	64.87	43205.00	< 0.0001
3	D-nnFraction	22.01	14656.74	< 0.0001
4	DE	17.71	11794.89	< 0.0001
5	BD	8.86	5901.77	< 0.0001
6	EJ	4.61	3068.80	< 0.0001
7	B-beta	4.46	2967.32	< 0.0001
8	AD	4.36	2904.26	< 0.0001
9	BE	3.84	2554.84	< 0.0001
10	H-problemSize	3.66	2439.84	< 0.0001
11	AJ	3.38	2250.48	< 0.0001
12	AB	3.26	2174.28	< 0.0001
13	CE	2.76	1836.57	< 0.0001
14	G-rhoLocal	2.57	1714.47	< 0.0001
15	D^2	2.45	1631.98	< 0.0001
16	DJ	2.31	1540.95	< 0.0001
17	AF	2.26	1504.79	< 0.0001
18	E^2	2.24	1492.39	< 0.0001
19	BJ	2.21	1468.81	< 0.0001
20	B^2	2.13	1417.47	< 0.0001
21	C-antsFraction	1.93	1286.26	< 0.0001
22	EG	1.88	1253.72	< 0.0001
23	CD	1.69	1125.00	< 0.0001
24	CJ	1.50	1001.96	< 0.0001
25	EF	1.30	862.88	< 0.0001
26	EH	1.23	820.42	< 0.0001
27	DG	0.98	649.57	< 0.0001
28	K-solutionConstruction	0.84	562.08	< 0.0001
29	AG	0.42	279.16	< 0.0001
30	CF	0.38	254.96	< 0.0001
31	H^2	0.32	215.21	< 0.0001
32	M-pheromoneUpdate	0.28	188.89	< 0.0001
33	G^2	0.26	172.66	< 0.0001
34	A^2	0.24	162.14	< 0.0001
35	FH	0.21	141.53	< 0.0001
36	EM	0.21	137.62	< 0.0001
37	F-rho	0.20	134.68	< 0.0001
38	BH	0.20	132.31	< 0.0001
39	DF	0.20	131.48	< 0.0001
40	GH	0.19	124.68	< 0.0001
41	F^2	0.18	122.13	< 0.0001
42	J^2	0.15	100.38	< 0.0001
43	AH	0.14	94.24	< 0.0001
44	A-alpha	0.12	80.01	< 0.0001
45	FJ	0.12	77.53	< 0.0001
46	EK	0.10	63.99	< 0.0001
47	HJ	0.09	58.21	< 0.0001
48	DK	0.06	42.16	< 0.0001
49	JK	0.06	42.03	< 0.0001
50	DH	0.06	41.72	< 0.0001
51	AE	0.06	37.78	< 0.0001
52	FM	0.05	33.60	< 0.0001
53	HK	0.05	32.45	< 0.0001
54	CG	0.04	28.05	< 0.0001
55	BM	0.04	25.03	< 0.0001
56	DM	0.04	25.02	< 0.0001
57	GK	0.04	24.44	< 0.0001
58	JM	0.03	22.86	< 0.0001
59	BC	0.03	22.76	< 0.0001
60	GJ	0.03	19.37	< 0.0001
61	CM	0.03	17.58	< 0.0001
62	BG	0.03	17.42	< 0.0001
63	CH	0.02	16.61	< 0.0001
64	FG	0.02	9.99	0.0016
65	KM	0.01	7.98	0.0048
66	GL	0.01	7.51	0.0062
67	BK	0.01	7.49	0.0062
68	GM	0.01	6.05	0.0140
69	EL	0.01	5.58	0.0183
70	L-antPlacement	0.01	4.39	0.0363
71	FL	0.00	2.92	0.0878
72	BF	0.00	2.75	0.0972

Fig. 11.5: Relative error–time ANOVA of relative error response from the full model. The table lists the remaining terms in the model after stepwise regression in order of decreasing sum of squares

Rank	Term	Sum of squares	F value	p value
1	C-antsFraction	1214.31	35326.80	< 0.0001
2	C^2	248.18	7219.98	< 0.0001
3	H-problemSize	122.02	3549.80	< 0.0001
4	D-nnFraction	31.19	907.44	< 0.0001
5	E-q0	4.88	141.91	< 0.0001
6	DH	4.57	132.87	< 0.0001
7	EM	3.37	98.14	< 0.0001
8	HJ	2.75	80.00	< 0.0001
9	K-solutionConstruction	2.59	75.23	< 0.0001
10	M-pheromoneUpdate	2.44	71.03	< 0.0001
11	EF	1.95	56.76	< 0.0001
12	BD	1.44	41.91	< 0.0001
13	DE	1.43	41.68	< 0.0001
14	GM	1.31	38.21	< 0.0001
15	DM	1.23	35.79	< 0.0001
16	CD	1.23	35.73	< 0.0001
17	EG	1.14	33.10	< 0.0001
18	CG	1.04	30.19	< 0.0001
19	BM	0.63	18.19	< 0.0001
20	CH	0.59	17.20	< 0.0001
21	CF	0.50	14.58	0.0001
22	JM	0.45	13.21	0.0003
23	FJ	0.43	12.41	0.0004
24	AE	0.42	12.36	0.0004
25	G-rhoLocal	0.35	10.30	0.0013
26	AB	0.33	9.68	0.0019
27	B-beta	0.31	9.09	0.0026
28	AG	0.30	8.64	0.0033
29	HK	0.27	7.79	0.0053
30	BE	0.25	7.34	0.0068
31	BC	0.24	7.08	0.0078
32	FM	0.24	6.88	0.0088
33	DK	0.22	6.29	0.0122
34	FH	0.19	5.39	0.0203
35	AH	0.18	5.31	0.0213
36	AJ	0.18	5.20	0.0227
37	GH	0.17	4.91	0.0268
38	BJ	0.17	4.86	0.0276
39	AC	0.15	4.29	0.0385
40	EJ	0.14	4.19	0.0408
41	J-problemStD	0.13	3.69	0.0547
42	BG	0.11	3.26	0.0712
43	AF	0.11	3.06	0.0804
44	HM	0.09	2.76	0.0966
45	A-alpha	0.01	0.28	0.5963
46	F-rho	0.01	0.22	0.6386

Fig. 11.6: Relative error–time ANOVA of time response from the full model. The table lists the remaining terms in the model after stepwise regression in order of decreasing sum of squares

lution quality and solution time simultaneously and so neglects this compromise. A technique from DOE allows multiple response models to be simultaneously tuned.

The desirability function approach (Montgomery 2005, Croarkin and Tobias 2006) is a widely used industrial method for optimizing multiple responses. The basic idea is that a process with many quality characteristics is completely unacceptable if any of those characteristics are outside some desired limits. For each response Y_i, a *desirability function* $d_i(Y_i)$ assigns a number between 0 and 1 to the possible values of the response Y_i; $d_i(Y_i) = 0$ is a completely undesirable value, and $d_i(Y_i) = 1$ is an ideal response value. These individual k desirabilities are combined into an overall desirability D using a geometric mean:

$$D = (d_1(Y_1) \times d_2(Y_2) \times \ldots \times d_k(Y_k))^{1/k} . \tag{11.1}$$

A particular class of desirability function was proposed by Derringer and Suich (1980). Let L_i and U_i be the lower and upper limits, respectively, of response i. Let T_i be the target value. If the target value is a maximum then

$$d_i = \begin{cases} 0 & y_i < L_i \\ \left(\frac{y_i - L_i}{T_i - L_i}\right)^r & L_i \leq y_i \leq T_i \\ 1 & y_i > T_i \end{cases} \tag{11.2}$$

If the target is a minimum value then

$$d_i = \begin{cases} 1 & y_i < T_i \\ \left(\frac{U_i - y_i}{U_i - T_i}\right)^r & L_i \leq y_i \leq T_i \\ 0 & y_i > U_i \end{cases} \tag{11.3}$$

The value r adjusts the shape of the desirability function. A value of $r = 1$ is linear. A value of $r > 1$ increases the emphasis of being close to the target value. A value of $0 < r < 1$ decreases this emphasis.

The multiple responses of *solution time* and *relative error* are expressed in terms of *desirability functions*. The *overall desirability* is then the geometric mean of the individual desirabilities. A numerical optimization is applied to the response surface models' equations such that the desirability is maximized. Typically, we specify the optimization in algorithm research with the dual goals of minimizing both solution error and solution time, while allowing all algorithm-related factors to vary within their design ranges.[4] Equal priority is given to the dual goals in this tutorial but these priorities can be varied. Recall that problem characteristics are also factors in the model since we want to establish the relationship between these problem charac-

[4] It is important to note that optimization of desirability does not necessarily lead to parameter recommendations that yield optimal algorithm performance. Desirability functions are a geometric mean of the desirability of each individual response. Furthermore, a response surface model is an interpolation of the responses from various points in the design space. There is therefore no guarantee that the recommended parameters result in optimal performance; they only result in tuned performance that is better than performance in most of the design space.

teristics, the algorithm parameters, and the performance responses. It does not make sense to include these problem characteristic factors in the optimization. The optimization process would naturally select the easiest problems as part of its solution. We therefore needed to choose fixed combinations of the problem characteristics and perform the numerical optimizations for each of these combinations. A sensible choice of such combinations is a three-level factorial of the characteristics, although any level factorial is possible depending on available resources. A more detailed description of these methods follows:

1. **Combinations of problem characteristics.** A three-level factorial combination of the problem characteristics is created. In the case of two characteristics, this creates 9 combinations of problem characteristics.
2. **Numerical optimization.** For each of these problem characteristic combinations, the Nelder-Mead simplex algorithm (Nelder and Mead 1965) was used for numerical optimization of overall desirability. The optimization goal is to minimize *both* the solution error response and the solution runtime response. The problem characteristics are fixed at the values corresponding to the 3 level factorial combinations.
3. **Choose the best solution.** When the optimization has completed, the solution of the parameter settings with the highest desirability is kept and the others are discarded. Note that there may be several solutions of very similar desirability but with differing factor settings. This is due to the nature of the multiobjective optimization and the possibility of many regions of interest.
4. **Further refine the solution.** Engineering judgement and experience may lead us to further refine the most desirable solution. For example, in this tutorial we will round off integer-valued parameters that are exponents to the nearest integer value. This is because exponents are expensive to compute and our pilot studies showed little gain in solution quality for this price.

This optimization procedure has recommended parameter settings for 9 locations covering the problem space defined in Table 11.7. Of course, a user requiring more refined parameter recommendations will have to run this optimization procedure for the problem characteristics of the scenario to hand. Optimization of desirability is done on the response surface equations. Other expensive algorithm runs beside those of the design points do not have to be executed.

The rankings of the ANOVA terms has already highlighted the factors that have little effect on the responses. For example, beta is always low, except when the problem standard deviation is high. The exploration/exploitation threshold q0 is always at a maximum of 0.99, implying that exploitation is always preferred to exploration. AntsFraction is always low. The remaining unimportant factors take on a variety of values in the model desirability optimization.

Table 11.7: Full relative error–time model results of desirability optimization. The table lists the recommended parameter values for combinations of problem size and problem standard deviation. The expected time and relative error are listed with the desirability value

Size	StDev	alpha	beta	antsFraction	nnFraction	q0	rho	rhoLocal	solutionConstruction	antPlacement	pheromoneUpdate	Time05	Relative Error	Desirability
300	10	8	2	1.00	1.00	0.99	0.69	0.96	parallel	random	bestSoFar	1.15	0.46	0.96
300	40	13	5	1.00	1.00	0.98	0.95	0.28	sequential	random	bestSoFar	1.46	1.24	0.86
300	70	1	11	1.00	20.00	0.98	0.05	0.70	parallel	random	bestSoFar	1.77	2.18	0.80
400	10	8	4	1.00	1.00	0.99	0.11	0.81	parallel	random	bestSoFar	2.42	0.46	0.92
400	40	13	6	2.19	1.16	0.97	0.99	0.03	parallel	random	bestOfItera	2.83	1.33	0.82
400	70	1	11	1.61	20.00	0.98	0.01	0.07	parallel	random	bestOfItera	4.92	2.59	0.73
500	10	7	3	1.13	1.00	0.99	0.86	0.01	parallel	same	bestOfItera	4.88	0.38	0.88
500	40	13	7	1.00	1.00	0.99	0.99	0.48	parallel	random	bestSoFar	4.25	1.35	0.80
500	70	1	10	1.04	19.78	0.99	0.05	0.01	parallel	same	bestOfItera	9.24	2.54	0.70

11.6.7 Discussion

The following conclusions are drawn from the ACS tuning study:

- **Unimportant factors: Ant placement not important.** The type of ant placement has no significant effect on ACS performance in terms of solution quality or solution time. **Alpha not important.** Alpha has no significant effect on ACS performance in terms of solution quality or solution time. This confirms the common recommendation in the literature of setting alpha equal to 1. **Rho not important.** Rho has no significant effect on ACS performance in terms of solution quality or solution time. This is a new result for ACS. **Pheromone Update Ant not important.** The ant used for pheromone updates is ranked highly for solution time.
- **Most important tuning parameters.** The most important ACS tuning parameters are the heuristic exponent B-beta, the number of ants C-antsFraction, the length of candidate lists D-nnFraction, the exploration/exploitation threshold E-q0, and rhoLocal.
- **Minimum order model.** A model that is of at least quadratic order is required to model ACS solution quality and ACS solution time. This is a new result for ACS and shows that a one-factor-at-a-time approach is not an appropriate way to tune the performance of ACS.
- **Relationship between tuning, problems, and performance.** The model of *relative error–time* was a good predictor of ACS performance across the entire design space.

11.6.8 Summary

The strengths of this chapter's methodologies come from the strengths of DOE. The methodologies are adapted from well established and tested methodologies used in other fields and are therefore proven on decades of scientific and industrial experience. The fractional factorial experiment designs provide a vast saving in experiment runs. Because DOE and response surface models build a model of performance across the whole design space, many research questions can be explored. Numerical optimization of this surface can quickly recommend tuning parameter settings for different weightings of the responses of interest. One may obtain settings appropriate for long runtimes and high quality or short runtimes and lower levels of solution quality. All of these questions are answered on the same model without the need to rerun experiments.

DOE is not a panacea for the myriad difficulties that arise in the empirical analysis of algorithms. Despite the efficiency of the DOE designs, running sufficient experiments to gather sufficient data is still computationally expensive. Of course, the experiments would have been orders of magnitude more expensive had a less sophisticated approach been used.

It is hoped that this chapter has convinced the reader of the merits of the DOE approach. The researcher who embraces these methodologies will have at their disposal an established, efficient, rigorous, reproducible approach for making strong conclusions about the relationship between algorithm tuning parameters, problem characteristics, and performance.

References

Applegate D, Bixby R, Chvatal V, Cook W (2003) Implementing the Dantzig-Fulkerson-Johnson algorithm for large traveling salesman problems. Mathematical Programming Series B 97(1-2):91–153

Croarkin C, Tobias P (eds) (2006) NIST/SEMATECH e-Handbook of Statistical Methods. National Institute of Standards and Technology, URL `http://www.itl.nist.gov/div898/handbook/`

Czarn A, MacNish C, Vijayan K, Turlach B, Gupta R (2004) Statistical Exploratory Analysis of Genetic Algorithms. IEEE Transactions on Evolutionary Computation 8(4):405–421

Derringer G, Suich R (1980) Simultaneous Optimization of Several Response Variables. Journal of Quality Technology 12(4):214–219

Dorigo M, Stützle T (2004) Ant Colony Optimization. The MIT Press, MA

Goldwasser M, Johnson DS, McGeoch CC (eds) (2002) Proceedings of the Fifth and Sixth DIMACS Implementation Challenges. American Mathematical Society

Lawler EL, Lenstra JK, Kan AHGR, Shmoys DB (eds) (1995) The Traveling Salesman Problem - A Guided Tour of Combinatorial Optimization. Wiley Series in Discrete Mathematics and Optimization, NY

Montgomery DC (2005) Design and Analysis of Experiments, 6th edn. Wiley
Nelder J, Mead R (1965) A simplex method for function minimization. The Computer Journal 7
Ostle B (1963) Statistics in Research, 2nd edn. Iowa State University Press
Rardin RL, Uzsoy R (2001) Experimental Evaluation of Heuristic Optimization Algorithms: A Tutorial. Journal of Heuristics 7(3):261–304
Ridge E (2007) Design of Experiments for the Tuning of Optimisation Algorithms. Phd thesis, Department of Computer Science, The University of York
Ridge E, Kudenko D (2007) Analyzing Heuristic Performance with Response Surface Models: Prediction, Optimization and Robustness. In: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, pp 150–157
Wu J, Hamada M (2000) Experiments: Planning, analysis, and parameter design optimization. Wiley, NY

Chapter 12
Using Entropy for Parameter Analysis of Evolutionary Algorithms

Selmar K. Smit and Agoston E. Eiben

Abstract Evolutionary algorithms (EA) form a rich class of stochastic search methods that share the basic principles of incrementally improving the quality of a set of candidate solutions by means of variation and selection (Eiben and Smith 2003, De Jong 2006). Such variation and selection operators often require parameters to be specified. Finding a good set of parameter values is a nontrivial problem in itself. Furthermore, some EA parameters are more relevant than others in the sense that choosing different values for them affects EA performance more than for the other parameters. In this chapter we explain the notion of entropy and discuss how entropy can disclose important information about EA parameters, in particular, about their relevance. We describe an algorithm that is able to estimate the entropy of EA parameters and we present a case study, based on extensive experimentation, to demonstrate the usefulness of this approach and some interesting insights that are gained.

12.1 Introduction and Background

Evolutionary algorithmsform a rich class of stochastic search methods that share the basic principles of incrementally improving the quality of a set of candidate solutions by means of variation and selection (Eiben and Smith 2003, De Jong 2006). Algorithms in this class are all based on the same generic framework (explained in the next section) and to obtain a concrete algorithm one needs to fill in many details, that is to say, specify the parameters of the algorithm. Over the history of EAs it has became clear that good parameter values are essential for good performance. However, as of today, not much is known about the effect of parameters on performance. Setting parameter values is commonly done in a very ad hoc manner,

Selmar K. Smit · Agoston E. Eiben
Vrije Universiteit Amsterdam, The Netherlands
e-mail: {sksmit, gusz}@cs.vu.nl

based on conventions, intuition, and experimental comparisons on a limited scale. Collective wisdom in evolutionary computing (EC) acknowledges that some parameters have more impact on performance than others. Obviously, more influential parameters need more care when setting their values, but at the moment there are no widely used techniques to establish the (relative) importance of different parameters. Screeningmethods (Ridge and Kudenko 2007b) are some of the few techniques currently used to indicate importance. However, the information that can be extracted is limited and the results of different algorithms cannot be compared.

In this chapter we show how entropycan be used to indicate how influential a particular parameter is. We use the term *parameter relevance*to reflect the level of influence on EA performance and argue that entropy is a good measure of relevance. The main contributions of this chapter are as follows:

1. We explain the notion of entropy and discuss how entropy can disclose important information on EA parameters, in particular, about their relevance.
2. We describe an algorithm, REVAC, that is able to estimate the entropy of EA parameters.
3. We present a case study, based on extensive experimentation, to demonstrate the usefulness of this approach and some interesting insights gained.

The rest of this chapter is organized as follows. In Sect. 12.2 we briefly introduce evolutionary algorithms, followed by a discussion on their parameters and issues in parameter tuning in Sect. 12.3. We elaborate on the notion of entropy in Sect. 12.4, including a discussion on the use of entropy for parameter analysis of EAs. The REVAC method is described in Sect. 12.5. Section 12.6 contains the case study and we conclude the paper in Sect. 12.7 by summarizing the main issues.

12.2 Evolutionary Algorithms

Evolutionary algorithms are all based on the same generic framework, inspired by biological evolution. The fundamental metaphor of evolutionary computing relates natural evolution to problem solving in a trial-and-error (a.k.a. generate-and-test) fashion, as illustrated in Table 12.1.

Table 12.1: The basic evolutionary computing metaphor linking natural evolution to problem solving

Evolution		Problem solving
environment	$\longleftrightarrow$	problem
individual	$\longleftrightarrow$	candidate solution
fitness	$\longleftrightarrow$	quality

In natural evolution, a given environment is filled with a population of individuals that strive for survival and reproduction. Their fitness—determined by the environment— tells how well they succeed in achieving these goals, i.e., it reflects their chances to live and multiply. In the context of problem solving we have a collection of candidate solutions. Their quality—defined by the given problem—determines the chance that they will be kept and used as seeds for constructing further candidate solutions.

Surprisingly enough, this idea of applying Darwinian principles to automated problem solving dates back to the 1940s, long before the breakthrough of computers (Fogel 1998). As early as in 1948 Turing proposed "genetical or evolutionary search" and already in 1962 Bremermann actually executed computer experiments on "optimization through evolution and recombination." During the 1960s three different implementations of the basic idea have been developed at three different places. In the USA Fogel et al. introduced evolutionary programming, (Fogel et al. 1966, Fogel 1995), while Holland called his method a genetic algorithm (Goldberg 1989, Holland 1992, Mitchell 1996). In Germany Rechenberg and Schwefel invented evolution strategies (Rechenberg 1973, Schwefel 1995). For about 15 years these areas developed separately; it is only since the early 1990s that they have been envisioned as different representatives of one technology that was termed evolutionary computing (Bäck 1996, Bäck et al. 2000a,b, Mitchell 1996). It was also in the early 1990s that a fourth stream following the general ideas has emerged: Koza's genetic programming (Banzhaf et al. 1998, Koza 1992). The contemporary terminology denotes the whole field as evolutionary computing and the methods therein as evolutionary algorithms. The historical versions evolutionary programming, evolution strategies, genetic algorithms, and genetic programming are seen as subtypes or dialects within the family of EAs.

As the history of the field suggests, there are many different variants of evolutionary algorithms. The common underlying idea behind all these techniques is the same. Given an objective function to be maximized we can randomly create a set of candidate solutions, i.e., elements of the objective function's domain that forms the search space, and apply the objective function as an abstract fitness measure—the higher the better. Based on this fitness, some of the better candidates are chosen to seed the next generation by applying so-called variation operators, recombination and mutation, to them. Recombination is a binary variation operator applied to two selected candidates (the so-called parents) and results in one or two new candidates (the children). Mutation is a unary variation operator, it is applied to one candidate and results in one new candidate. Executing recombination and mutation leads to a set of new candidates (the offspring) that compete—based on their fitness—with the old ones for a place in the next generation. This cycle can be iterated until a solution is found or a previously set computational limit is reached.

In this process there are two fundamental forces that form the basis of all evolutionary systems:

- *Variation* (implemented through recombination and mutation operators) creates the necessary diversity within the population, thus it facilitates novelty.

- *Selection* (implemented through parent selection and survivor selection operators) acts as a force towards increasing the quality of solutions in the population.

The combined application of variation and selectiongenerally leads to improving fitness values in consecutive populations. It is easy, although somewhat misleading, to view this process as if evolution is optimizing (or at least "approximizing") the fitness function, by approaching the optimal values closer and closer over time.

It should be noted that many components of such an evolutionary process are stochastic. Thus, although during selection fitter individuals have a higher chance of being selected than less fit ones, typically even the weak individuals have a chance of becoming a parent or of surviving. During the recombination process, the choice of which pieces from the parents will be recombined is made at random. Similarly for mutation, the choice of which pieces will be changed within a candidate solution, and of the new pieces to replace them, is made randomly.

It is easy to see that EAs fall into the category of generate-and-test algorithms. The fitness function represents a heuristic estimation of solution quality, and the search process is driven by the variation and selection operators. Evolutionary algorithms possess a number of features that can help to position them among generate-and-test methods:

- EAs are population based, i.e., they process a whole collection of candidate solutions simultaneously.
- EAs mostly use recombination, mixing information from two or more candidate solutions to create a new one.
- EAs are stochastic.

The various dialects of evolutionary computing that we have mentioned previously all follow the general EA outline, differing only in their technical details. In particular, the representation of a candidate solution is often used to characterize different streams. Typically the representation (i.e., the data structure encoding a candidate solution) has the form of strings over a finite alphabet in genetic algorithms (GAs), real-valued vectors in evolution strategies (ESs), finite state machines in classical evolutionary programming (EP), and trees in genetic programming (GP). The origin of these differences is mainly historical. Technically, one representation might be preferable to others if it matches the given problem better, that is, if it makes the encoding of candidate solutions easier or more natural. For instance, when solving a satisfiability problem with n logical variables, the straightforward choice is to use bit-strings of length n, hence the appropriate EA would be a genetic algorithm. To evolve a computer program that can play checkers, trees are well-suited (namely, the parse trees of the syntactic expressions forming the programs), thus a GP approach is likely. It is important to note that the recombination and mutation operators working on candidates must match the given representation. Thus, for instance, in GP the recombination operator works on trees, while in GAs it operates on strings. In contrast to variation operators, the selection process only takes fitness information into account, and so it works independently of the choice of representation. Therefore differences between the selection mechanisms commonly applied in each stream are a matter of tradition rather than of technical necessity.

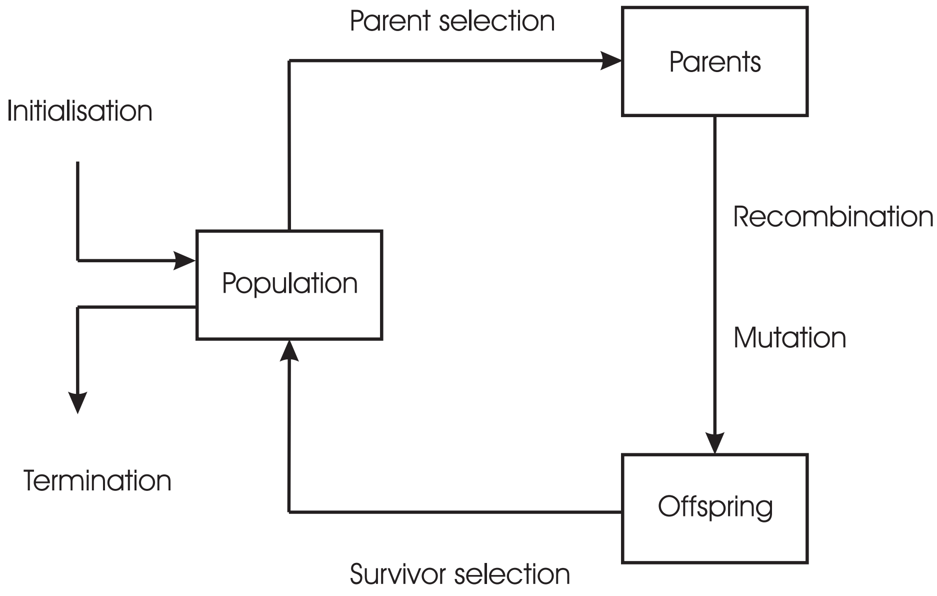


Fig. 12.1: The general scheme of an evolutionary algorithm as a flowchart

It is worth noting that the borders between the four main EC streams have been becoming less distint in the last decade. Approaching EAs from a "unionist" perspective the distinguishing features of different EAs are the algorithmic components, representation, recombination operator, mutation operator, parent selection operator, and survivor selection operator. Reviewing the details of the commonly used operators for these components exceeds the scope of this chapter. For these details we refer to a modern text book, such as Eiben and Smith (2003) or De Jong (2006), and in the sequel we will use (the names of) such operators without further explanation. Here we restrict ourselves to providing an illustration in Table 12.2, showing how particular choices can lead to a typical genetic algorithm or evolution strategy.

12.3 EA Design, EA Parameters

Given a particular problem, designing an EA for solving it requires filling in the details of the generic EA framework appropriately. For a solid basis we first elaborate on suitable naming conventions regarding these details.

One possibility is to call these details *EA parameters*. In this case, designing an EA for a given application amounts to selecting good values for these parameters. For instance, the definition of an EA might include setting the parameter

Table 12.2: A typical GA and ES as an instantiation of the generic EA scheme by particular representation and operators

	GA	ES
Representation	bit-strings	real-valued vectors
Recombination	1-point crossover	intermediary
Mutation	bit-flip	Gaussian noise by $N(0, \sigma)$
Parent selection	2-tournament	uniform random
Survivor selection	generational	(μ, λ)
Extra	none	self-adaptation of σ

`crossoveroperator` to `1-point`, the parameter `crossoverrate` to 0.5, and the parameter `populationsize` to 100. In principle, this is a sound naming convention, but intuitively, there is a difference between choosing a good crossover operator from a given list of three operators and choosing a good value for the related crossover rate $p_c \in [0,1]$. One feels that the parameters `crossoveroperator` and `crossoverrate` are different.

This difference can be formalized if we distinguish parameters by their domains. The parameter `crossoveroperator` has a finite domain with no sensible distance metric, e.g., $\{\texttt{1-point}, \texttt{uniform}, \texttt{averaging}\}$, whereas the domain of the parameter p_c is a subset of $\mathbb{R}$ with the natural metric for real numbers. This difference is essential for searchability. For parameters with a domain that has a distance metric, one can use heuristic search and optimization methods to find optimal values. For the first type of parameters this is not possible because the domain has no exploitable structure. The only option in this case is sampling. For a clear distinction between these cases we can use the terms *symbolic parameter* or *qualitative parameter*, e.g., `crossoveroperator`, and *numeric parameter* or *quantitative parameter*, e.g., crossover rate. For both types of parameters the elements of the parameter's domain are called *parameter values* and we instantiate a parameter by allocating a value to it.

An alternative naming convention (used in Nannen et al. (2008)), is to call symbolic parameters *components* and the elements of their domains *operators*. In the corresponding terminology a parameter is instantiated by a value, while a component is instantiated by allocating an operator to it. Using this naming convention for the example in the beginning of this section, `crossoveroperator` is a component instantiated by the operator `1-point`, while `crossoverrate` is a parameter instantiated by the value 0.5.

In this paper we adhere to the second terminology distinguishing components and parameters. Further to this, we distinguish two levels in designing a particular EA instance for a given problem by saying that the operators (the high-level, symbolic details) define the EA, while the parameters (the low-level, numerical details) define a variant of this EA. Table 12.3 illustrates this matter.

This terminology enables precise formulations, while enforcing care with phrasing. From now on the phrase *an EA for problem X* means a partially specified algo-

Table 12.3: Three EA instances specified by the components recombination, mutation, parent selection, survivor selection and the parameters mutation rate (p_c), mutation step size (σ), crossover rate (p_c), population size (μ), offspring size (λ), and tournament size. The EA instances in columns A and B are just variants of the same EA. The EA instance in column C belongs to a different EA.

	A	B	C
Recombination	1-point	1-point	averaging
Mutation	bit-flip	bit-flip	Gaussian $N(0,\sigma)$
Parent selection	tournament	tournament	uniform random
Survivor selection	generational	generational	(μ,λ)
p_m	0.01	0.1	0.05
σ	n.a.	n.a	0.1
p_c	0.5	0.7	0.7
μ	100	100	10
λ	n.a.	n.a	70
Tournament size	2	4	n.a

rithm where the operators to instantiate EA components are defined, but the parameter values are not. After specifying all details, including the values for all parameters, we obtain *an EA instance for problem X*.

It has long been noticed that EA parameters have a strong influence on EA performance. The problem of setting EA parameters correctly is therefore highly relevant. Setting EA parameters is commonly divided into two cases; parameter tuning and parameter control (Eiben et al. 1999). In case of parameter control the parameter values are changing during an EA run. In this case one needs initial parameter values and suitable control strategies, which in turn can be deterministic, adaptive, or self-adaptive. Parameter tuning is easier in the sense that the parameter values are not changing during a run, hence only a single value per parameter is required. Nevertheless, even the problem of tuning an EA for a given application is hard because there is a large number of options, but only little knowledge about the effect of EA parameters on EA performance. EA users mostly rely on conventions (mutation rate should be low), ad hoc choices (why not use population size 100), and experimental comparisons on a limited scale (testing combinations of three different crossover rates and three different mutation rates).

In these terms we can express the primary focus of this chapter as being parameter tuning. Entropy is proposed as a generic measure of parameter relevance that shows how difficult it is to find parameter values that induce good EA performance. The practical use of this information is obvious. Given an EA (thus, all operators specified), if the relevance levels of the parameters are known then it is possible to allocate tuning efforts such that more relevant parameters are tuned more extensively than less relevant ones. It is important to note that relevance information of EA parameters depends on two other factors: the EA itself (that is, the chosen operators to instantiate EA components) and the problem at hand. The aspect of problem dependence belongs to the issue of scoping, that is, establishing the scope of validity of experimental work. A thorough treatment of this issue exceeds the limitations of this chapter; our case study attempts to cope with this problem by using many problem instances produced by a parameterized random problem instance generator (see Sect. 12.6 for details). Concerning the dependence on the EA itself, the case study will illustrate that the approach we advocate here is also helpful for determining good operators for EA components, and thus for designing EAs in general. The basis of such aggregation is the hierarchy between EA components and EA parameters. This hierarchy is visible in Table 12.8, which arranges parameters by the operators to which they (with population size as the only exception). Relying on this hierarchy, it is possible to aggregate results concerning parameters to results at the level of operators and thus at the level of EAs.

12.4 Shannon and Differential Entropy

As we have seen, an EA can be composed of a wide variety of operators, each with its own numeric parameters that need properly chosen values for satisfactory EA performance. However, choosing proper values, i.e., tuning, requires effort both in terms of time and computing facilities, and both resources are limited in practice. Hence, tuning efforts should be carefully allocated to different parameters such that the most relevant parameters receive the most attention and little effort is spent on finding good values for parameter with limited relevance. The problem is how to quantify the relevance of parameters.

12.4.1 Using Success Ranges for Relevance Estimation

In order to objectively quantify a parameter's relevance, and thus the amount of tuning it needs, one can look at how accurately its value needs to be specified for achieving a given performance level. A straightforward approach is to measure which part of the parameter's range leads to the desired performance. For example, let us assume that an EA has two parameters, $X_1 \in [0,1]$ and $X_2 \in [0,1]$, and that the algorithm reaches some desired performance if X_1 is in the range [0, 0.5] **and** X_2 is in the range [0, 0.1]. One can argue that X_2 is more relevant, because X_1 has a success range of 50$\%$ of its full range [0, 1] and X_2 has a success range of 10$\%$. Furthermore, one can assume that the success range within the full 2D parameter space $[0,1] \times [0,1]$ is $50\% \cdot 10\% = 5\%$. However, this kind of reasoning can lead to misleading conclusions if the parameters are not independent. For example, if we have an algorithm that achieves the desired performance if X_1 is in the range $[0,0.5]$ **or** X_2 is in the range of [0, 0.1]. The success range of X_1 is in that case equal to $[0,1]$, because the EA instances with parameter values of $\langle 0,0.05 \rangle$ and $\langle 1,0.05 \rangle$ both terminate with success. Similarly, the success range of X_2 is equal to $[0,1]$ too.

12.4.2 Shannon Entropy

Entropy is commonly used to measure the amount of disorder of a system, and this concept has been extended in information theory to quantify the uncertainty associated with a random variable. To be precise, the (Shannon) entropy $H(X)$ of a random variable X with probability mass function $p(x)$ can be used to measure the average information content that is missing when the value of X is unknown (Shannon 1948). Shannon defined the entropy for discrete variables as

$$H(X) = -\sum_{x \in D_X} p(x) \cdot \log_2 p(x) \tag{12.1}$$

where D_X is the domain of X, $p(x)$ is equal to the chance of observing value x, and $\log_2$ is the logarithm with base 2.

Notice that a random variable with a large range of different values will have a higher entropy than a random variable with just a few specific values. For example, a fair coin has an entropy of 1 bit. A biased coin has an entropy that is lower, because it will return one side more often than the other. Predicting the next value for a biased coin is easier, lowering the uncertainty. So, entropy can be seen as a measure of the extent of bias towards a certain value, or range of values.

12.4.3 Using the Shannon Entropy for Relevance Estimation

Let us consider an evolutionary algorithm with two parameters, population size $P \in \{10, 100\}$ and tournament size $T \in \{5, 10\}$. We can now execute the EA 100 times with all possible parameter value combinations and thus experimentally establish whether a given combination is successful. Success here can mean that the EA finds the optimal fitness value in all runs, or that the mean best fitness (MBF) over all runs is above a certain threshold. Table 12.4 shows a possible outcome.

Table 12.4: Success or failure for different parameter value combinations, 1 = success, 0 = failure

		Population size	
		10	**100**
Tournament size	**5**	1	1
	10	0	1

We can observe that a high population size (100), a low tournament size (5), or a combination of both, leads to success. The list of population sizes that lead to success will therefore be {10, 100, 100} with p values of $\frac{1}{3}$ and $\frac{2}{3}$. The entropy of this distribution is therefore $\frac{1}{3} \cdot \log_2(\frac{1}{3}) + \frac{2}{3} \cdot \log_2(\frac{2}{3}) = 0.92$ bits. Unlike the success range-based measure from Sect. 12.4.1, the entropy identifies correctly that there is a bias for one of those two values. Thereby it indicates that it is beneficial to choose the parameter value from a specific area, rather than selecting an arbitrary value. The size of the entropy indicates the size of the area: The lower the entropy, the smaller the area that leads to success.

Furthermore, we can use this approach to show how the desired performance is related to the required tuning effort. To this end we need a fine-graded overview of the experimental outcomes that exhibits the mean best fitness over the 100 EA runs belonging to the parameter values used in those runs. Table 12.5 shows a possible outcome for five different population sizes and fixed tournament size (not shown in the table).

Table 12.5: Mean best fitness for different population sizes

Population size	**Performance (MBF)**
10	0.80
20	0.85
30	0.90
40	0.95
50	1.00

Based on these results, we can calculate the entropy not only for single parameter values, but for a whole range of values. This results in a table containing the desired performance, and the corresponding entropy (Table 12.6).

Table 12.6: The minimal performance required for success and the corresponding entropy

Performance (MBF) required for success	**Entropy**
0.80	2.32
0.85	2.00
0.90	1.59
0.95	1.00
1.00	0.00

We can use such a table or graph to determine the size of the set of all possible parameter values that lead to the desired performance. Furthermore, this can indicate how relevant it is to tune a certain parameter with a specific minimal performance in mind. In this case, each of the five population sizes leads to a performance of at least 0.8. The entropy, using a minimal performance of 0.8, is therefore the highest. If we define success as reaching a performance of at least 1.0, then only one setup (population size = 50) results in success. The corresponding entropy is therefore the lowest. In terms of (un)certainty, if we observe success in this case, then we know for sure that the EA used population size 50. While observing success in the first case we do not know anything, because all possible population sizes could have caused it.

12.4.4 Differential Entropy

The differential entropy is an extension of the Shannon entropy to the domain of continuous probability distributions. This is required for parameters that are real-valued, for example mutation rate. It is clear that calculating the entropy of such parameters requires a somewhat different approach than enumerating all possible combinations of parameter values.

One approach is to divide the continuous domain into a certain number of bins. Because this makes the domain discrete, we can use the Shannon entropy as described in the previous section. However, the number of bins highly influences the outcomes. One way of dealing with this problem is always using the same number of bins. This makes the results comparable, but could lead to problems if the number of bins is too small. The best number of bins would therefore be infinity, which is exactly the approach that is used with the differential entropy.

In order to calculate the differential entropy, it is required that the probability distribution of such a parameter is known. Just as with the Shannon entropy, this can be any distribution. With probability density function $f(x)$, the entropy is defined as

$$h(X) = -\int_{\mathbb{X}} f(x) \log_2 f(x)\, dx \tag{12.2}$$

Unlike the Shannon entropy, the differential entropy can get negative. For example, a uniform distribution over the range $[0, 0.1]$ results in a differential entropy of:

$$\begin{aligned}
f(x) &= \frac{1}{0.1-0} && (12.3)\\
h(X) &= -\int_0^{0.1} f(x) \log_2 f(x)\, dx && (12.4)\\
&= \log_2(0.1) && (12.5)\\
&= -3.3 && (12.6)
\end{aligned}$$

In order to compare the entropy of distributions that are defined over different parameter ranges in a meaningful way, we normalize the range of all parameters to the unit interval $[0, 1]$ before calculating the entropy. In this way the uniform distribution has a Shannon entropy of zero, and any other distribution has a negative Shannon entropy.

12.4.5 Joint Entropy

The notion of entropy as introduced above can be calculated for each specific parameter. However, sometimes we are interested in the entropy of constructs that depend on more then one parameters. For instance, the Gaussian (p, σ) mutation operator is regulated by the mutation probability p and the mutation stepsize σ. The amount of tuning required by this mutation operator will thus depend on the amount of tuning required by two parameters. This idea can also be extended to the level of the algorithm, namely the set of all instantiated operators. For an illustration recall Table 12.3 and observe that the EA with the instances in columns A and B depends on four parameters and the EA whose instance is shown in column C depends on five. In such situations we need to handle the combination of more parameters, that is, to calculate joint entropies. If we assume independence of the parameters in question, then the joint entropy is equal to the sum of the individual entropies. If the parameters are not independent then one should calculate the combined probability density function and use this to calculate the entropy. If this is not possible, one can use lower and upper bounds for the joint entropy that are easy to calculate, because the sum of the individual entropies forms the lower bound and the maximum individual entropy is the upper bound of the joint entropy.

$$h(X \cap Y) \leq \quad h(X) + h(Y) \tag{12.7}$$

$$h(X \cap Y) \geq \max(h(X), h(Y)) \tag{12.8}$$

To illustrate the usage of such bounds assume that we need information on the relevance of the uniform crossover operator (one parameter, p_c) and the Gaussian (p_m, σ) mutation operator. Assume furthermore that the entropies belonging to p_c, p_m, and σ are known. Then the sum of entropies of the parameters p_m and σ is an upper bound for the entropy of the mutation operator (that would correspond to the joint entropy of p_m and σ). Thus, if the sum of entropies of the parameters p_m and σ is lower than the entropy of p_c, then we know that the entropy of this mutation operator is lower than the entropy of this crossover operator. Application to complete EAs with more parameters is similar.

12.5 Estimating Entropy

Calculating the entropy as proposed in the previous section is a computationally intensive task. Even if one performs a full parameter sweep over thousands of different parameters settings, the resulting entropy is still just an estimation. Although such a sweep can be distributed over multiple machines (Samples et al. 2007), it is still a very time-consuming task, especially because much time is spent on evaluating parameter settings that are not interesting, because their performance is far from optimal.

There are three different approaches to estimate the entropy more efficiently. Ranking and selection of parameters can be used to estimate entropy (Branke et al. 2005) with less effort. The principle is not very different from a full parameter sweep. However, instead of assigning each parameter setting the same computational effort, ranking and selection focuses on the areas with a high utility. This results in a better estimated entropy, and expectedly, a higher level of utility with the same computational effort.

Secondly, one could build models of the utility landscapeand calculate the entropy through the model. There are several approaches to create such models, for example, sequential parameter optimization (Bartz-Beielstein 2003, 2006) and response surface models(Ridge and Kudenko 2007a). Some of those models can directly be translated into a probability density function, for which the entropy is given by (12.2). In other cases, a sweep over all possible parameter values can be used to calculate the entropy of the model. Because utility is estimated by the model, rather than tested, the entropy can be estimated efficiently.

Finally, one could use heuristic search methods that iteratively generate parameter vectors to be tested and used to calculate entropy. The search heuristic should represent a bias towards better parameter vectors, thus allocating more computational efforts to interesting areas of the search space. Because of this bias, the estimations of entropy will be better in high-utility regions, quite similarly to ranking and selection methods. At this moment we only know of one method in this category: REVAC (relevance estimation and value calibration) (Nannen and Eiben 2007a,b). REVAC has been developed to aid the design of evolutionary mechanisms for simulation and optimization in application areas without much knowledge on successful EA designs (Nannen and Eiben 2006). The main activities of REVAC can be summarized as follows. Given a problem to be solved and an EA to solve it with

- REVAC finds parameter vectors with high utility;
- REVAC collects values of entropy for different utility levels;
- REVAC creates a distribution for each parameter that indicates the expected utility of parameter values.

It is important to note that REVAC does not handle parameter interactions (no joint distributions for multiple parameters) and that it can be used for tuning numeric parameters only.

The case study, described in Sect. 12.6, is based on estimating entropy values. In principle, the experiments could have been conducted using any of the methods mentioned above, but actually we have used REVAC (Nannen et al. 2008). Therefore, we describe it in detail in the sequel.

12.5.1 REVAC: the Algorithm

Technically, REVAC is a heuristic generate-and-test method that is iteratively adapting a set of parameter vectors of a given EA. Testing a parameter vector is done by executing the EA with the given parameters and measuring the EA performance. EA performance can be defined by any appropriate performance measure, or combination of performance measures, and the results will reflect the utility of the parameter vector in question. Because of the stochastic nature of EAs, in general a number of runs is advisable to obtain better statistics.

For a good understanding of the REVAC method it is helpful to distinguish two views on a given set of parameter vectors as shown in Table 12.7. Taking a *horizontal* view on the table, each row shows the name of a vector (first column), the k parameter values of this vector, and the utility of this vector (last column), defined through the performance of the EA in question. However, taking a *vertical* view on the table, the ith column in the inner box shows m values from the domain of parameter i and this can be seen as a distribution over the range of that parameter.

Table 12.7: Two views on a table of parameter vectors

	$\mathcal{D}(x_1)$	$\cdots$	$\mathcal{D}(x_i)$	$\cdots$	$\mathcal{D}(x_k)$	Utility
$\mathbf{x}^1$	$\{x_1^1$	$\cdots$	x_i^1	$\cdots$	$x_k^1\}$	u^1
$\vdots$		$\ddots$				$\vdots$
$\mathbf{x}^n$	$\{x_1^n$	$\cdots$	x_i^n	$\cdots$	$x_k^n\}$	u^n
$\vdots$				$\ddots$		$\vdots$
$\mathbf{x}^m$	$\{x_1^m$	$\cdots$	x_i^m	$\cdots$	$x_k^m\}$	u^m

To understand how REVAC generates parameter vectors the horizontal view is more helpful. From this perspective, REVAC can be described as an evolutionary algorithm, in the style of EDAs (Mühlenbein and Höns 2005), working on a population of m parameter vectors. This population is updated by selecting parent vectors, which are then recombined and mutated to produce one child vector that is then inserted into the population. The exact details are as follows:

- **Parent selection** is deterministic in REVAC as the best n ($n < m$) vectors of the population, i.e., those with the highest utility, are selected to become the parents of the new child vector. For further discussion we denote the set of parents by $\{\mathbf{y}^1, \ldots, \mathbf{y}^n\} \subset \{\mathbf{x}^1, \ldots, \mathbf{x}^m\}$.
- **Recombination** is performed by a multiparent crossover operator, uniform scanning. In general, this operator can be applied to any number of parent vectors and the ith value in the child $\langle c_1, \ldots, c_k \rangle$ is selected uniformly random from the ith values, $y_i^1, \ldots, y_i^n$, of the parents. Here we create one child from the selected n parents.
- **Mutation**, applied to the offspring $\langle c_1, \ldots, c_k \rangle$ created by recombination, works independently on each parameter $i \in \{1, \ldots, k\}$ in two steps. First, a

mutation interval $[a_i, b_i]$ is calculated, then a random value is chosen uniformly from this interval. The mutation interval for a given c_i is determined by all values $y_i^1, \ldots, y_i^n$ for this parameter in the selected parents as follows. First, the parental values are sorted in increasing order such that $y_i^1 \leq \cdots \leq y_i^n$. (Note that, for the sake of readability, we do not introduce new indices corresponding to this ordering.) Recall that the child $\langle c_1, \ldots, c_k \rangle$ is created by uniform scanning crossover, hence the value c_i comes from one of the parents. That is, $c_i = y_i^j$ for some $j \in \{1, \ldots, n\}$ and we can define the neighbors of c_i as follows. The first neighbors of c_i are y_i^{j-1} and y_i^{j+1}, the second neighbors are y_i^{j-2} and y_i^{j+2}, the third neighbors are y_i^{j-3} and y_i^{j+3}, etc. Now, the begin point a_i of the mutation interval is defined as the hth lower neighbor of c_i, while the end point of the interval b_i is the hth upper neighbor of c_i, where h is a parameter of the REVAC method (as there are no neighbors beyond the upper and lower limits of the domain, we extend it by mirroring the parent values as well as the mutated values at the limits). The mutated value c_i' is drawn from this mutation interval $[a_i, b_i]$ with a uniform distribution and the child $\langle c_1', \ldots, c_k' \rangle$ is composed from these mutated values.

- **Survivor selection** is also deterministic in REVAC as the newly generated vector always replaces the oldest vector in the population.
- **Evaluation** The newly generated vector is tested by running the EA in question with the values it contains.

The above list describes one REVAC cycle that is iterated until the maximum number of vectors tested is reached.

12.5.2 REVAC: the Data Generated

In each REVAC cycle several data records are saved to allow analysis after termination. This happens directly after the n parent vectors are selected from the population. First, the lowest utility in the set of parents is identified as $u = \min\{u^1, \ldots, u^n\}$. Then for each parameter $i \in \{1, \ldots, k\}$ we calculate the entropy e_i and store the pair $\langle e_i, u \rangle$. The calculation of e_i is based on the set $\{y_i^1, \cdots, y_i^n\}$ of parental values for parameter i that we consider to be a representative sample of good parameter values, "good" defined as leading to a utility higher than u.[1]

For the calculation of e_i we use the formula for differential entropy (Equation 12.2) applied to the probability density function

$$f^i(z) = \frac{1}{(n+1) \cdot (b(z) - a(z))} \quad , \tag{12.9}$$

where $a(z)$ and $b(z)$ are the hth lower and upper neighbor of z, respectively.

[1] In a previous study (Nannen et al. 2008) we associated the entropy e_i with the expected utility $v = \text{avg}\{u^1, \ldots, u^n\}$ among the parents. In other words, we defined "good" as leading to an expected utility v.

For determining the (hth) neighbors of any given z, we use a method similar to the procedure for defining the neighbors of c_i in the description of the mutation operator. However, in general, there need not be a $j \in \{1,\dots,n\}$ such that $z = y_i^j$. Hence, the index j is now defined as the one satisfying $y_i^j \leq z < y_i^{j+1}$ and we call y_i^j and y_i^{j+1} the first neighbors of z, y_i^{j-1} and y_i^{j+2} its second neighbors, y_i^{j-2} and y_i^{j+3} its third neighbors, etc. Now, $a(z)$ and $b(z)$ in (12.9) are the hth lower neighbor and the hth upper neighbor of z, respectively .

Calculating the entropy e_i for all $i = 1,\dots,k$ we get k pairs, $\langle e_1, u\rangle$ through $\langle e_k, u\rangle$, one for each parameter. These pairs can be used for making plots of performance levels (parameter vector utilities) and entropy values.

12.6 Case Study

Table 12.8: EA components, operators, and parameters used in this study

Component	**Operator**	**Parameter(s)**
		population size μ
Parent selection	tournament	(parent) tournament size
	random uniform	-
	fitness proportional	-
	best selection	number n of best
Survivor selection	generational	-
	tournament	(survivor) tournament size
	random uniform	-
	(μ, λ)	λ
	$(\mu + \lambda)$	λ
Recombination	none	-
	one-point	crossover probability
	uniform	crossover probability
Mutation	reset	mutation probability
	Gaussian($\sigma, 1$)	step size
	Gaussian(σ, p)	step size, mutation probability

In this section we present a case study, based on data generated by a large experimental investigation (Nannen April, 2009, Nannen et al. 2008). Our case study will present entropy data that is inherently produced by every REVAC run. Strictly speaking, the use of REVAC-generated data implies that we are not showing results on entropy, but results on the estimation of entropy as done by REVAC. Our discussion, however, will be in general terms since it could be presented along the same lines with any other similar method for entropy estimation.

12.6.1 Experimental Setup

For a clear discussion we separate three different levels that can be distinguished in the context of algorithm design.

1. The problem/application (here: fitness landscapes created by a problem generator)
2. The problem solver (here: an evolutionary algorithm)
3. The design method for calibrating the problem solver (here: REVAC)

To obtain concrete problem instances to be solved by the EAs we use a parameterized random problem instance generator that produces real-valued fitness landscapes or objective functions to be maximized. This generator (Gallagher and Yuan 2006) defines a class of landscapes formed by the max-set of Gaussian curves in high dimensional Cartesian spaces. Where a Gaussian mixture model takes the average of several Gaussians, a max-set takes their enveloping maximum. In this way, the complexity of maximizing a Gaussian mixture can be combined with full control over the location and height of global and local maxima. For this study we selected problem set 4 from (Gallagher and Yuan 2006) with peaks that get higher the closer they get to the origin. Using 10 dimensions, 100 Gaussians and the same distributions over height, location, and rotation of these Gaussians we generated 10 test landscapes by different random seeds.

For the EA we use the open source Evolutionary Computation toolkit in Java (ECJ)[2] which allows the specification of a full EA through a simple parameter file. Obviously, we do not use all possibilities ECJ offers, but select a number of operators for the EA components and run REVAC for all those EAs that can be obtained by the combinations of these operators. To be specific, we base our study on the components parent selection, survivor selection, recombination, and mutation, with three to five commonly used operators for each, as shown in Table 12.8. We follow the naming convention of ECJ. For any given EA, the population size parameter is always present; other parameters depend on the actual chosen operators. Due to technical details in ECJ, only 10 different combinations of parent and survivor selection operators are possible.[3] Together with 3 choices for the recombination operator and 3 choices for the mutation operator, this yields 90 different EAs to be tuned, of which 6 EAs with 2, 27 with 3, 38 with 4, 17 with 5, and 2 with 6 free parameters. Most operators have one or no parameter to calibrate. One operator has 2 parameters: Gaussian(σ, p) with free parameters σ for step size and p for mutation probability, the latter set to one in the case of Gaussian(σ, 1).

The basic data nuggets in this case study are produced by REVAC runs with a given EA on one of the 10 test landscapes. In one run REVAC is allowed to generate and test 1,000 parameter vectors. Generating parameter vectors is done through the

[2] A Java based Evolutionary Computation Research System, `http://www.cs.gmu.edu/~eclab/projects/ecj/`

[3] Arguably, (μ, λ) and $(\mu + \lambda)$ define both parent *and* survivor selection. Here we classify them under survivor selection because that is what the parameter λ influences.

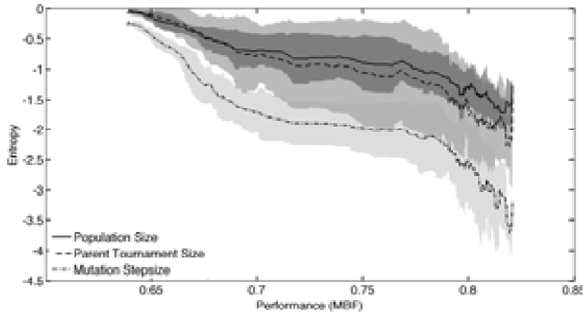


Fig. 12.2: Parameter entropy plot for EA-1: {Tournament Parent Selection, Generational Survivor Selection, No Crossover and Gaussian(σ)}

main REVAC loop using $m = 100$ vectors to form the population and selecting the best $n = 50$ of them as parents to create one child vector by uniform scanning crossover. REVAC's smoothing parameter used in the mutation operator is set at $h = 5$. Testing parameter vectors happens by executing 10 independent runs of the given EA using the given vector. The utility of a vector is measured by the performance of the EA using that vector, which is in turn measured by the mean best fitness, that is, the best fitness value after each run, averaged over the 10 independent runs.

The experimental data used in this section are generated by carrying out 10 REVAC runs with all of the 90 EAs on all of the 10 test landscapes. The basic data points are pairs of estimated entropy values and corresponding performance levels, as explained in Sect. 12.5.2. All together we have 901 of such pairs per REVAC run (because, after initialization, 900 generations have passed before REVAC terminates), hence the plots shown in the sequel are based on $10 \times 901 = 9010$ pairs.

12.6.2 Entropy of Parameters

Each of the 90 EAs can be individually analyzed to gain insight into the relevance of its parameters. For example, let EA-1 be the evolutionary algorithm defined by the operators {Tournament Parent Selection, Generational Survivor Selection, No Crossover and Gaussian(σ)} for its components. EA-1 has three parameters, namely population size, tournament size, and mutation stepsize (σ). For each of these parameters we can plot the entropy for reaching a specific performance using our REVAC-generated data.

Figure 12.2 shows the mean and standard deviation of the estimated entropy for a given performance. From this figure it is clear that mutation stepsize is the most relevant parameter, and the other two parameters do not differ significantly w.r.t. their relevance. For example, if the desired performance is equal to 0.75, the entropy of stepsize is equal to -2, while the population size and tournament size have an entropy higher than -1. This means that the mutation stepsize has the smallest range of values that lead to a performance of at least 0.75. Thus, for this EA it is advisable to dedicate the most tuning effort to the stepsize parameter.

Furthermore, we can observe a rather straight line of the average entropy of stepsize between approximately 0.7 and 0.78. This indicates that the size of the area that leads to values of at least 0.7 is equal to the size of the area that leads to a performance of 0.78 or higher. Therefore, we can conclude that it is equally hard to tune the EA to reach a performance of 0.7 or of 0.78. However, for a higher performance, not only stepsize, but also the other parameters need to be carefully set.

12.6.3 Entropy of Operators

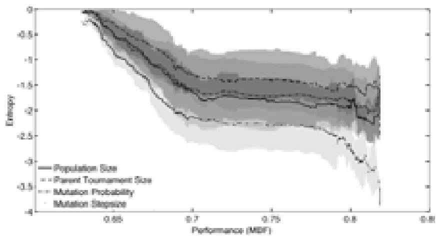


Fig. 12.3: Parameter entropy plot for EA-2: {Tournament Parent Selection, Generational Survivor Selection, No Crossover and Gaussian(p, σ)}

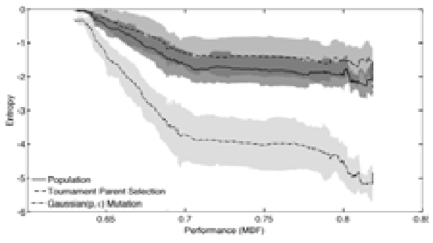


Fig. 12.4: Operator entropy plot for EA-2: {Tournament Parent Selection, Generational Survivor Selection, No Crossover and Gaussian(p, σ)} (different scale w.r.t. Fig. 12.3)

In general, operators can have zero, one, or more parameters. For the following example we use an EA that has an operator with two parameters, EA-2, defined by {Tournament Parent Selection, Generational Survivor Selection, No Crossover and Gaussian(p, σ)}. The Gaussian(p, σ) mutation operator is an extension of the Gaussian(σ) operator, where the probability p of applying the mutation operator is a parameter. (The usual option of always applying mutation is now a special case belonging to $p = 1$.) As we will show, a parameter entropy plot similar to Fig. 12.2 could lead to incorrect conclusions in this case.

Looking at Fig. 12.3 we can observe that the entropy levels of the parameters do not differ very much, all being somewhere between -1.5 and -2.5 for a range of performance levels between 0.7 and 0.8. This implies that these *parameters* require roughly equal tuning effort. However, one can not infer from this figure that the mutation *operator* requires roughly the same tuning effort as the other details of the algorithm. Because the Gaussian(p,σ) operator has two parameters, both parameters have to be taken into account. As explained in Sect. 12.4.5, the joint entropy of two parameters can be estimated by the sum of both entropies. This leads to Fig. 12.4, where entropy data is elevated from parameter level to operator level, except for population size, of course.[4] Note that for tournament selection (and in general for all operators with just one parameter) the parameter-level and the operator-level plots are identical. This figure clearly shows that Gaussian(p,σ) is by far the most relevant operator for EA-2, requiring the most tuning effort.

12.6.4 Entropy of EAs

The previous examples illustrated matters within one given EA. In particular, we showed that comparison of entropies can provide valuable information about which design detail (parameter or operator) is the most relevant and thus requires the most tuning effort. Because this all happened in the context of one EA, the operators were only meant to instantiate different EA components (within the given EA). In this section we consider information about different operators for the same component (thus leading to different EAs). We can distinguish these two cases by the main question that can be answered by looking at the data plots. In the previous subsections the question was: Given an EA, which parameter, respectively operator, of this EA needs to be tuned most carefully for a given level of desired performance?, whereas here we address the following question: Given all operators for instantiating an EA component, which of these operators implies the most effort for tuning?

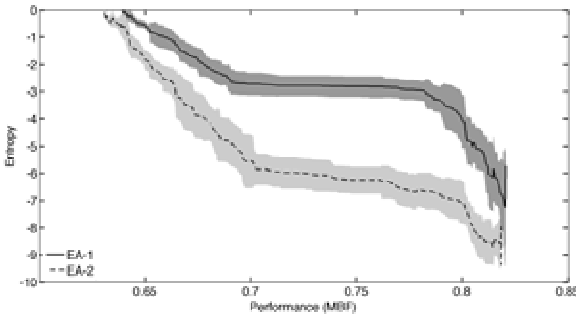


Fig. 12.5: Algorithm entropy plot for EA-1 (with the Gaussian(σ) mutation operator) and EA-2 (with the Gaussian(p,σ) mutation operator)

Posing this question concerning mutation operators, we might try to get an answer from the previous figures on the average entropies of the Gaussian(σ) and Gaussian(p,σ) operators. The entropy levels are around -3 and -5 for reaching a performance of 0.8, respectively. This indicates that the Gaussian(p,σ) mutation operator requires more tuning effort to reach the same performance than the

[4] Recall that EA-2 does not use an operator for the crossover component and that the Survivor selection component is instantiated through "generational," which has no parameters.

Gaussian(σ) operator. However, this would be too hasty a conclusion because it ignores the influence of other operators and parameters. It is therefore important to compare the total entropy of an algorithm design.

Figure 12.5 shows these algorithm entropies for EA-1 and EA-2. The curves show that the Gaussian(σ) operator indeed causes a higher algorithm entropy, and therefore probably requires less tuning. The difference at the algorithm level is even larger than we observed in Fig. 12.4 and Fig. 12.2 at the operator level.

A similar comparison can be done for the crossover component. To this end, we define EA-3 as {Tournament Parent Selection, Generational Survivor Selection, One Point Crossover, and Gaussian(σ) Mutation} and compare it with EA-1, wchich used no crossover at all. The plots are shown in Fig. 12.7. Obviously, EA-1 has a crossover operator entropy of 0 at all performance levels, while the crossover operator entropy of EA-3 is most likely lower than 0. Therefore, one would expect that the algorithm entropy of EA-3 is lower than that of EA-1. However, the absence of crossover makes that variation is only regulated by a single parameter. This causes the range of possible σ values that lead to success strongly decreases if the required performance increases. In this case, the entropy of σ decreases significantly (Fig. 12.7). The total algorithm entropy of the algorithm without crossover is therefore lower than the algorithm entropy of the algorithm with one-point crossover (Fig. 12.6). This indicates that adding crossover to the algorithm decreases tuning effort.

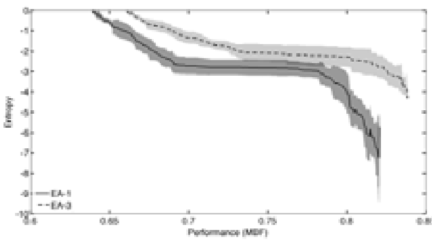


Fig. 12.6: Algorithm entropy plot for EA-1 (without crossover) and EA-3 (with crossover)

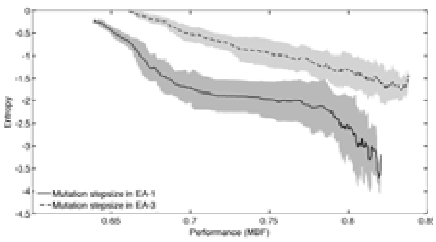


Fig. 12.7: Mutation stepsize entropy plot for EA-1 (without crossover) and EA-3 (with crossover)

12.7 Conclusions

In this chapter we illustrated how entropy can be used to obtain useful information on evolutionary algorithm parameters. The main problem we consider here is the tuning of EA parameters, that is, the process of searching for good parameter values in the space of all possibilities. The related challenges are rooted in the facts that 1) EA parameters need to be instantiated with good values for good EA performance,

2) some EA parameters are more relevant than others in the sense that choosing different values for them affects EA performance more than for those other parameters, and 3) to maximize the effectivity of parameter tuning, tuning efforts should be allocated such that more relevant parameters are given more time/capacity to find good values than less relevant parameters.

As we explained, entropy (two versions discussed) is a good indicator of parameter relevance. Furthermore, we described the REVAC method (Nannen April, 2009) that can be used to collect data on specific performance levels and corresponding entropy values. Using a large data set generated by many REVAC runs we created plots concerning the (estimated) entropy of EA parameters, EA operators, and complete EAs as well. For the sake of correctness, let us recall that the entropy values in the source data set are calculated from a pool of parameter vectors as known to REVAC at a given moment, that the performance level associated with an entropy value is the utility of the worst vector in the pool, that the combined entropies in our plots are estimated by the sum of the underlying entropies, and that our data were based on just one problem (albeit more random instances of it). With these caveats in mind we presented a number of case studies to illustrate the kind of knowledge that can be gained through the entropy-based approach. These case studies showed that it is indeed possible to distinguish parameters and operators that need more tuning than others and that differences in relevance can be quantified. Our entropy-based analysis also disclosed a formerly unknown advantage of using crossover in an EA: It reduces the tuning effort associated with mutation. This kind of knowledge has immediate practical relevance in guiding users when tuning their EAs and more theoretical advantages as it can disclose relationships between parameters in general. The scope of validity of such insights can be small (the set of problem instances used in the experiments), medium range (more problems of a certain type), or rather generic ("all" EAs and "all" problems). Although the case studies here serve mainly as illustrations, not as crisp claims, we do believe that many of our findings hold in more cases than just the 10 landscapes and the 90 EAs we used here and that more such findings are possible. This, however, requires more experimental research in the future.

Acknowledgement

All experimental data and results are due to Volker Nannen.

References

Bäck T (1996) Evolutionary Algorithms in Theory and Practice. Oxford University Press, Oxford, UK

Bäck T, Fogel D, Michalewicz Z (eds) (2000a) Evolutionary Computation 1: Basic Algorithms and Operators. Institute of Physics Publishing, Bristol

Bäck T, Fogel D, Michalewicz Z (eds) (2000b) Evolutionary Computation 2: Advanced Algorithms and Operators. Institute of Physics Publishing, Bristol

Banzhaf W, Nordin P, Keller R, Francone F (1998) Genetic Programming: An Introduction. Morgan Kaufmann, San Francisco CA

Bartz-Beielstein T (2003) Experimental Analysis of Evolution Strategies: Overview and Comprehensive Introduction. Tech. Rep. Reihe CI 157/03, SFB 531, Universität Dortmund, Dortmund, Germany, URL `http://sfbci.informatik.uni-dortmund.de/home/English/`

Bartz-Beielstein T (2006) Experimental Research in Evolutionary Computation—The New Experimentalism. Natural Computing Series, Springer

Branke J, Chick SE, Schmidt C (2005) New developments in ranking and selection: an empirical comparison of the three main approaches. In: Proceedings of the 37th Winter Simulation Conference (WSC 2005), Winter Simulation Conference, pp 708–717

De Jong K (2006) Evolutionary Computation: A Unified Approach. The MIT Press

Eiben A, Smith J (2003) Introduction to Evolutionary Computation. Natural Computing Series, Springer

Eiben A, Hinterding R, Michalewicz Z (1999) Parameter Control in Evolutionary Algorithms. IEEE Transactions on Evolutionary Computation 3(2):124–141

Fogel D (1995) Evolutionary Computation. IEEE Press

Fogel D (ed) (1998) Evolutionary Computation: the Fossil Record. IEEE Press, Piscataway, NJ

Fogel L, Owens A, Walsh M (1966) Artificial Intelligence through Simulated Evolution. Wiley, Chichester, UK

Gallagher M, Yuan B (2006) A general-purpose tunable landscape editor. IEEE Transactions on Evolutionary Computation 10(5):590–603

Goldberg D (1989) Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley

Holland J (1992) Adaption in Natural and Artificial Systems. MIT Press, Cambridge, MA, 1st ed: 1975, The University of Michigan Press, Ann Arbor

Koza J (1992) Genetic Programming. MIT Press, Cambridge, MA

Lipson H (ed) (2007) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2007), ACM

Mitchell M (1996) An Introduction to Genetic Algorithms. MIT Press, Cambridge, MA

Mühlenbein H, Höns R (2005) The estimation of distributions and the minimum relative entropy principle. Evolutionary Computation 13(1):1–27

Nannen V (April, 2009) Evolutionary agent-based policy analysis in dynamic environments. PhD thesis, Vrije Universiteit Amsterdam

Nannen V, Eiben A (2006) A method for parameter calibration and relevance estimation in evolutionary algorithms. In: Keijzer M (ed) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2006), ACM, pp 183–190

Nannen V, Eiben AE (2007a) Efficient Relevance Estimation and Value Calibration of Evolutionary Algorithm Parameters. In: IEEE Congress on Evolutionary Computation, IEEE, pp 103–110

Nannen V, Eiben AE (2007b) Relevance Estimation and Value Calibration of Evolutionary Algorithm Parameters. In: Veloso MM (ed) IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, pp 1034–1039

Nannen V, Smit S, Eiben A (2008) Costs and benefits of tuning parameters of evolutionary algorithms. In: Rudolph G, Jansen T, Lucas SM, Poloni C, Beume N (eds) PPSN, Springer, Lecture Notes in Computer Science, vol 5199, pp 528–538

Rechenberg I (1973) Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution. Fromman-Hozlboog Verlag, Stuttgart

Ridge E, Kudenko D (2007a) Analyzing heuristic performance with response surface models: prediction, optimization and robustness. In: Lipson (2007), pp 150–157

Ridge E, Kudenko D (2007b) Screening the parameters affecting heuristic performance. In: Lipson (2007), pp 180–180

Samples M, Byom M, Daida J (2007) Parameter sweeps for exploring parameter spaces of genetic and evolutionary algorithms. In: Lobo F, Lima C, Michalewicz Z (eds) Parameter Setting in Evolutionary Algorithms, Springer, pp 161–184

Schwefel HP (1995) Evolution and Optimum Seeking. Wiley, New York NY

Shannon C (1948) A mathematical theory of communication. Bell System Technical Journal 27:379–423, 623–656

Chapter 13
F−Race and Iterated F−Race: An Overview

Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle

Abstract Algorithms for solving hard optimization problems typically have several parameters that need to be set appropriately such that some aspect of performance is optimized. In this chapter, we review `F−Race`, a racing algorithm for the task of automatic algorithm configuration. `F−Race` is based on a statistical approach for selecting the best configuration out of a set of candidate configurations under stochastic evaluations. We review the ideas underlying this technique and discuss an extension of the initial `F−Race` algorithm, which leads to a family of algorithms that we call iterated `F−Race`. Experimental results comparing one specific implementation of iterated `F−Race` to the original `F−Race` algorithm confirm the potential of this family of algorithms.

13.1 Introduction

Many state-of-the-art algorithms for tackling computationally hard problems have a number of parameters that influence their search behavior. Such algorithms include exact algorithms such as branch-and-bound algorithms, algorithm packages for integer programming, and approximate algorithms such as stochastic local search (SLS) algorithms. The parameters can roughly be classified into numerical and categorical parameters. Examples of numerical parameters are the tabu tenure in tabu search algorithms or the pheromone evaporation rate in ant colony optmization (ACO) algorithms. Additionally, many algorithms can be seen as being composed of a set of specific components that are often interchangeable. Examples are different branching strategies in branch-and-bound algorithms, different types of crossover operators in evolutionary algorithms, and different types of local search algorithms in

Mauro Birattari · Zhi Yuan · Prasanna Balaprakash · Thomas Stützle
IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium
e-mail: {mbiro,zyuan,pbalapra,stuetzle}@ulb.ac.be

iterated local search. These interchangable components are often well described as categorical parameters of the underlying search method.

Research has clearly shown that the performance of parameterized algorithms depends strongly on the particular values of the parameters, and the choice of an appropriate setting of these parameters is itself a difficult optimization problem (Adenso-Diaz and Laguna 2006, Birattari 2009, Birattari et al. 2002). Given that typically not only the setting of numerical parameters but also that of categorical parameters needs to be determined, we call this problem also the *algorithm configuration problem*. An important aspect of this problem is that it is typically a stochastic problem. In fact, there are two main sources of stochasticity. The first is that often the algorithm itself is stochastic because it uses some randomized decisions during the search. In fact, this stochasticity is typical for SLS algorithms (Hoos and Stützle 2004). However, even if an algorithm is deterministic, its performance and search behavior depend on the particular instance to which it is applied. In fact, the particular instance being tackled can be seen as having been drawn according to some underlying, possibly unknown probability distribution, introducing in this way a second stochastic factor.

In our research, we have developed a method, called `F-Race`, which is particularly well suited for dealing with this stochastic aspect. It is a method that is inspired from racing algorithms in machine learning, in particular Hoeffding races (Maron 1994, Maron and Moore 1994, 1997). The essential idea of racing methods, in general, and ours in particular, is to evaluate a given set of candidate configurations iteratively on a stream of instances. As soon as enough statistical evidence is gathered against some candidate configurations, these are eliminated and the race continues only with the surviving ones. In our case, this method uses after each evaluation round of the candidate configurations the nonparametric Friedman test as a family-wise test: it checks whether there is evidence that at least one of the configurations is significantly different from others. If the null hypothesis of no differences is rejected, Friedman post-tests are applied to eliminate those candidate configurations that are significantly worse than the best one.

In this chapter, we first formally describe the algorithm configuration problem, following Birattari et al. (2002) and Birattari (2009). Next, in Sect. 13.3, we give details on `F-Race`. Section 13.4 discusses considerations on the sampling of candidate configurations, proposes a family of iterated `F-Race` algorithms, and defines one specific iterated `F-Race` algorithm, which extends over an earlier version published by Balaprakash et al. (2007). Computational results with this new variant, which are presented in Sect. 13.5, confirm its advantage over other ways of generating the candidate configurations for `F-Race`. We end the chapter with an overview of available `F-Race` applications and outline ideas for further research.

13.2 The Algorithm Configuration Problem

F-Race is a method for offline configuration of parameterized algorithms. In the *training phase* of offline tuning, an algorithm configuration is to be determined in a limited amount of time that optimizes some measure of algorithm performance. The final algorithm configuration is then deployed in a *production phase* where the algorithm is used to solve previously unseen instances.

A crucial aspect of this *algorithm configuration problem* is that it is a problem of generalization, as occurs in other fields such as machine learning. Based on a given set of training instances, the goal is to find high-perfoming algorithm configurations that perform well on (a potentially infinite set of) unseen instances that are not available when deciding on the algorithm's parameters. Hence, one assumption that is tacitly made is that the set of training instances is representative of the instances the algorithm faces once it is employed in the production phase. The notions of best performance, generalization, etc. are made explicit in the formal definition of the algorithm configuration problem.

13.2.1 The Algorithm Configuration Problem

The problem of configuring a parameterized algorithm can be formally defined as a 7-tuple $\langle \Theta, I, p_I, p_C, t, \mathcal{C}, T \rangle$, where:

- Θ is the possibly infinite set of candidate configurations.
- I is the possibly infinite set of instances.
- p_I is a probability measure over the set I.
- $t : I \to \mathbb{R}$ is a function associating to every instance the computation time that is allocated to it.
- $c(\theta, i, t(i))$ is a random variable representing the cost measure of a configuration $\theta \in \Theta$ on instance $i \in I$ when run for computation time $t(i)$.[1]
- $C \subset \mathbb{R}$ is the range of c, that is, the possible values for the cost measure of the configuration $\theta \in \Theta$ on an instance $i \in I$.
- p_C is a probability measure over the set C: With the notation $p_C(c|\theta, i)$ we indicate the probability that c is the cost of running configuration θ on instance i.
- $\mathcal{C}(\theta) = \mathcal{C}(\theta|\Theta, I, p_I, p_C, t)$ is the criterion that needs to be optimized with respect to θ. In the most general case it measures in some sense the desirability of θ.

[1] To make the notation lighter, in the following we often will not mention the dependence of the cost measure on $t(i)$. We use the term *cost* to refer, without loss of generality, to the minimization of some performance measure such as the objective function value in a minimization problem or the computation time taken for a decision problem instance.

- T is the total amount of time available for experimenting with the given candidate configurations on the available instances before delivering the selected configuration.[2]

On the basis of these concepts, solving the problem of configuring a parameterized algorithm is to find the configuration $\bar{\theta}$ such that

$$\bar{\theta} = \arg\min_{\theta \in \Theta} \mathcal{C}(\theta). \tag{13.1}$$

Throughout the whole chapter, we consider for $\mathcal{C}$ the expected value of the cost measure c

$$\mathcal{C}(\theta) = E_{I,C}[c] = \int c \, \mathrm{d}p_C(c|\theta, i) \, \mathrm{d}p_I(i), \tag{13.2}$$

where the expectation is considered with respect to both p_I and p_C, and the integration is taken in the Lebesgue sense (Billingsley 1986). However, other options for defining the cost measure to be minimized such as the median cost or a percentile of the cost distribution are easily conceivable.

The measures p_I and p_C are usually not explicitly available and the analytical solution of the integrals in (13.2) is not possible. In order to overcome this limitation, the expected cost can be estimated in a Monte Carlo fashion on the basis of running the particular algorithm configuration on a training set of instances.

The cost measure c in (13.2) can be defined in various ways. For example, the cost of a configuration θ on an instance i can be measured by the objective function value of the best solution found in a given computation time $t(i)$. In such a case, the task is to tune algorithms for an optimization problem and the goal is to optimize the solution quality reached within a given computation time. In the case of decision problems, the goal is rather to choose parameter settings such that the computation time to arrive at a decision is minimized. In this case, the cost measure would be the computation time taken by an algorithm configuration to decide on an instance i. Since arriving at a decision may take infeasibly long computation times, the role played by the function t is to give a maximum computation time budget for the execution of the algorithm configuration. If after a cutoff time of $t(i)$ the algorithm has not finished, the cost measure may use additional penalties (Hutter et al. 2007). Finally, let us remark that the definition of the algorithm configuration problem applies not only to the configuration of stochastic algorithms, but it extends also to deterministic, parameterized algorithm: in this case, $c(\theta, i, t(i))$ is strictly speaking no longer a random variable but a deterministic function; the stochasticity is then due to the instance distribution p_I.

One basic question concerns how many times a configuration should be evaluated on each of the available problem instances for estimating the expected cost. Assuming that the performance of a stochastic algorithm is evaluated by a total of N runs, it has been proved by Birattari (2004a, 2009) that sampling N instances

[2] In the following, we refer to T also as *computational budget*; often it will be measured as the number of algorithm runs instead of a total amount of computation time.

with one run on each instance results in the lowest variance of the estimator. Hence, it is always preferable to have a large set of training instances available. If, however, only a few training instances are provided, one needs to go back to evaluating algorithm configurations on the instances more than once.

13.2.2 Types of Parameters

As said in the introduction, algorithms can have different types of parameters. There we distinguished between *categorical* and *numerical* parameters. Categorical parameters typically refer to different procedures or discrete choices that can be taken by an algorithm (or, more generally, an algorithm framework such as a metaheuristic). In SLS algorithms examples are the type of perturbations and the particular local search algorithm used in iterated local search (ILS) or the type of neighborhood structure to be used in iterative improvement algorithms. Sometimes it is possible to order the categories of these categorical parameter according to some surrogate measure. For example, neighborhoods may be ordered according to their size, or crossover operators in genetic algorithms according to the disruption they introduce. Hence, sometimes categorical parameters can be converted into ordinal ones. (We are, however, not aware of configuration methods that exploited this possibility so far.) Categorical parameters that may be ordered based on secondary criteria, we call `pseudo-ordinal` parameters.[3]

Besides categorical parameters, numerical parameters are common in many algorithms. `Continuous` numerical parameters take as values some subset of the real numbers. Examples of these are the pheromone evaporation rate in ACO, or the cooling rate in simulated annealing. Often, numerical parameters take integer values; an example is the strength of a perturbation that is measured by the number of solution components that change. If such parameters have a relatively large domain, they may be treated in the configuration task as continuous parameters, which are then rounded to the next integer. In the following we call such integer parameters `quasi-continuous` parameters.

Furthermore, it is often the case that some parameter is only in effect when another parameter, usually a categorical one, takes certain values. This is the case of a `conditional` parameter. An example can be given in ILS, where as one option a tabu search may be used as the local search; in this case, the tabu list length parameter is a conditional parameter that depends on whether a categorical parameter "type of local search" indicates that tabu search is used.[4] The `F-Race`-based configura-

[3] Note that, strictly speaking, binary parameters are also ordinal ones, although they are usually handled without considering an ordering.

[4] It is worth noticing that sometimes it may make sense to replace a numerical parameter by a categorical parameter plus a conditional parameter, if changing the numerical parameter may lead to drastic changes in design choices of an algorithm. Consider as an example the probability of applying a crossover operator. This parameter may take a value of zero, which indicates actually that no crossover is applied. In such cases it may be useful to introduce a binary parameter, which

tion algorithms described in this chapter are able to handle all the aforementioned types of parameters, including conditional parameters.

13.3 `F-Race`

The conceptually simplest approach for estimating the expected cost of an algorithm configuration θ, as defined by (13.2), is to run the algorithm using a sufficiently large number of instances. This estimation can be repeated for a number of candidate configurations and, once the overall computational budget allocated for the selection process is consumed, the candidate configuration with the lowest estimate is chosen as the best performing configuration. This is an example of what can be characterized as the *brute-force approach* to algorithm configuration.

There are two main problems associated with this brute-force approach. The first is that one needs to determine a priori how often a candidate configuration is evaluated. The second is that also poor performing candidate configurations are evaluated with the same amount of computational resources as the good ones.

13.3.1 The Racing Approach

As one possibility to avoid the disadvantages of the brute-force approach we have used a racing approach. The racing approach originated from the machine learning community (Maron and Moore 1994), where it was first proposed for solving the model selection problem (Burnham and Anderson 2002). We adapted this approach to make it suitable for the algorithm configuration task. The racing approach performs the evaluation of a finite set of candidate configurations using a systematic way to allocate the computational resources among them. The racing algorithm evaluates a given finite set of candidate configurations `step` by `step`. At each `step`, all the remaining candidate configurations are evaluated in parallel,[5] and the poor candidate configurations are discarded as soon as sufficient statistical evidence is gathered against them. The elimination of the poor candidates allows to focus the computations on the most promising ones to obtain lower variance estimates for these. In this way, the racing approach overcomes the two major drawbacks of the brute-force approach. First, it does not require a fixed number of steps for each candidate configuration but it determines it adaptively based on statistical evidence. Second, poor performing candidates will not be evaluated as soon as enough evi-

indicates whether crossover is used or not, together with a conditional parameter on the crossover probability, which is only used if the binary parameter indicates that crossover is used.

[5] A round of function evaluations of surviving candidate configurations on a certain instance is called an `evaluation step`, or simply a `step`. By `function evaluation`, we refer to one run of the candidate configuration on one instance.

dence is gathered against them. A graphical illustration of the *racing* algorithm and the *brute-force* approach is shown in Fig. 13.1.

To describe the racing approach formally, suppose a sequence of training instances i_k, with $k = 1, 2, \ldots$, is randomly generated from the target class of instances I following the probability model p_I. Denote by c_k^θ the cost of a single run of a candidate configuration θ on instance i_k. The evaluation of the candidate configurations is performed incrementally such that at the kth `step`, the array of observations for evaluating θ,

$$\mathsf{c}^k(\theta) = \left(c_1^\theta, c_2^\theta, \ldots, c_k^\theta\right),$$

is obtained by appending c_k^θ to the end of the array $\mathsf{c}^{k-1}(\theta)$. A *racing* algorithm then generates a sequence of nested sets of candidate configurations

$$\Theta_0 \supseteq \Theta_1 \supseteq \Theta_2 \supseteq \ldots,$$

where Θ_k is the set of the surviving candidate configurations after `step` k. The sets of surviving candidate configurations start from a finite set $\Theta_0 \subseteq \Theta$, which is typically obtained by sampling $|\Theta_0|$ candidate configurations from Θ. How the initial set of candidate configurations can be generated is the topic of Sect. 13.4. The step from a set Θ_{k-1} to Θ_k is obtained by possibly discarding some configurations that appear to be suboptimal on the basis of information that becomes available at `step` k.

At `step` k, when the set of the surviving candidates is Θ_{k-1}, a new instance i_k is considered. Each candidate $\theta \in \Theta_{k-1}$ is tested on i_k and each observed cost c_k^θ is appended to the respective array $\mathsf{c}^{k-1}(\theta)$ to form the arrays $\mathsf{c}^k(\theta)$ for each $\theta \in \Theta_{k-1}$. `Step` k terminates, defining set Θ_k by dropping from Θ_{k-1} the candidate configurations that appear to be suboptimal based on some statistical test that compares the arrays $\mathsf{c}^k(\theta)$ for all $\theta \in \Theta_{k-1}$.

The above described procedure is iterated and stops either when all candidate configurations but one are discarded, a given maximum number of instances have been sampled, or when the predefined computational budget B has been exhausted.[6]

13.3.2 The Peculiarity of F-Race

F-Race is a racing algorithm based on the nonparametric Friedman's two-way analysis of variance by ranks (Conover 1999), for short, Friedman test. This algorithm was first proposed by Birattari et al. (2002) and studied in detail in Birattari's PhD thesis (Birattari 2004b).

[6] The computational budget may be measured as a total available computation time T (see the definition of the configuration problem on page 313). It is, however, often more convenient to define the maximum number of function evaluations, if each function evaluation is limited to the same amount of computation time.

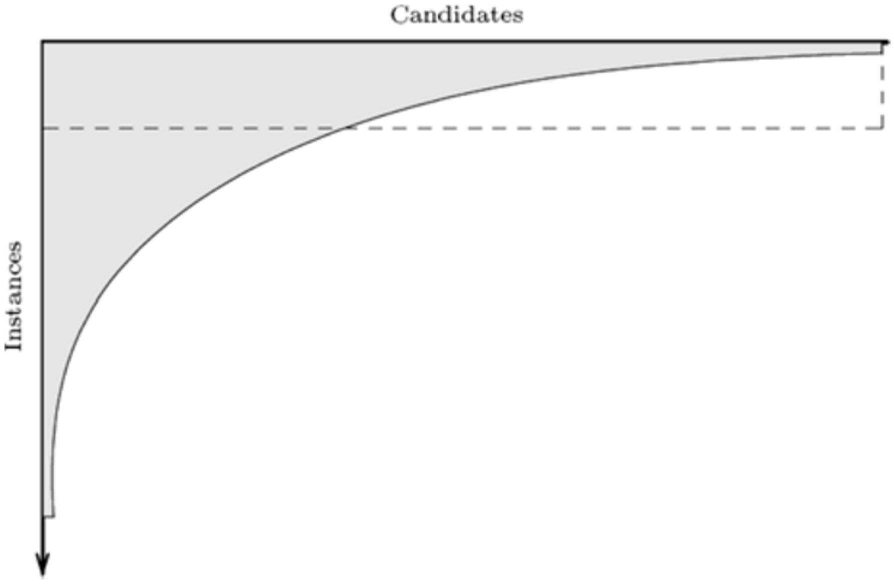


Fig. 13.1: Graphical representation of the allocation of configuration evaluations by the *racing* approach and the *brute-force* approach. In the *racing* approach, as soon as sufficient evidence is gathered that a candidate is suboptimal, such a candidate is discarded from further evaluation. As the evaluation proceeds, the *racing* approach thus focuses more and more on the most promising candidates. On the other hand, the *brute-force* approach tests all given candidates on the same number of instances. The shadowed figure represents the computation performed by the *racing* approach, while the dashed rectangle the one of the *brute-force* approach. The two figures cover the same surface, that is, the two approaches are allowed to perform the same total number of experiments

To describe `F-Race`, assume it has reached `step` k, and $m = |\Theta_{k-1}|$ candidate configurations are still in the race. The Friedman test assumes that the observed costs are k mutually independent m-variate random variables

$$\begin{array}{ccccc}
b_1 = \Big(c_1^{\theta_{v_1}}, & c_1^{\theta_{v_2}}, & \ldots, & c_1^{\theta_{v_m}}\Big) \\
b_2 = \Big(c_2^{\theta_{v_1}}, & c_2^{\theta_{v_2}}, & \ldots, & c_2^{\theta_{v_m}}\Big) \\
\vdots \qquad \vdots & \vdots & \ddots & \vdots \\
b_k = \Big(c_k^{\theta_{v_1}}, & c_k^{\theta_{v_2}}, & \ldots, & c_k^{\theta_{v_m}}\Big)
\end{array}$$

called blocks, where each block b_l corresponds to the computational results obtained on instance i_l by each surviving configuration at `step` k.

Within each block, the costs c_l^θ are ranked in nondecreasing order; average ranks are used in case of ties. For each configuration $\theta_{v_j} \in \Theta_{k-1}$, R_{lj} is the rank of θ_{v_j}

in block b_l, and $R_j = \sum_{l=1}^{k} R_{lj}$ is the sum of ranks for configuration θ_{v_j}, over all instances i_l, with $1 \leq l \leq k$. The test statistic used by the Friedman test is the following (Conover 1999):

$$T = \frac{(m-1)\sum_{j=1}^{m}\left(R_j - \frac{k(m+1)}{2}\right)^2}{\sum_{l=1}^{k}\sum_{j=1}^{m} R_{lj}^2 - \frac{km(m+1)^2}{4}}.$$

Under the null hypothesis that all candidates are equivalent, T is approximately χ^2 distributed with $m-1$ degrees of freedom (Papoulis 1991). If the observed value of T is larger than the $1-\alpha$ quantile of this distribution, the null hypothesis is rejected. This indicates that at least one candidate configuration gives better performance than at least one of the others.

If the null hypothesis is rejected in this *family-wise* test, it is justified to do pairwise comparisons between individual candidates. There are various ways of conducting these Friedman *post hoc tests*. For F-Race, we have chosen one particular one that is presented in the book of Conover (1999): candidates θ_j and θ_h are considered to be statistically significantly different if

$$\frac{|R_j - R_h|}{\sqrt{\frac{2k\left(1-\frac{T}{k(m-1)}\right)\left(\sum_{l=1}^{k}\sum_{j=1}^{m} R_{lj}^2 - \frac{km(m+1)^2}{4}\right)}{(k-1)(m-1)}}} > t_{1-\alpha/2},$$

where $t_{1-\alpha/2}$ is the $1-\alpha/2$ quantile of the Student's t distribution.

If F-Race does not reject at step k the null hypothesis of the family-wise comparison, all candidate configurations in Θ_{k-1} pass to Θ_k; if the null hypothesis is rejected, pairwise comparisons are performed between the best candidate configuration and each other one. The best candidate configuration is selected as the one that has the lowest expected rank. All candidate configurations that result significantly worse than the best one are discarded and will not appear in Θ_k.

When only two candidates remain in the race, the Friedman test reduces to the *binomial sign test for two dependent samples* (Sheskin 2000). However, in the F-Race algorithm, the *Wilcoxon matched-pairs signed-ranks test* (Conover 1999) is adopted, for the reason that the Wilcoxon test is more powerful and data efficient than the binomial sign test in such a case (Siegel and Castellan 1988).

In F-Race, the test statistic is based on the ranking of the candidates. Ranking plays an important twofold role. The first one is due to the nonparametric nature of a test based on ranking. A second role played by ranking in F-Race is to implement in a natural way a blocking design (Dean and Voss 1999, Montgomery 2000). By focusing only on the ranking of the different configurations within each instance, this blocking design becomes an effective way of normalizing the costs observed on different instances.

13.4 The Sampling Strategy for `F-Race`

In the previous section, the question of how the set of candidate configurations Θ_0 is defined was left open. This is the question we address in this section; in fact, it should be clear that this question is rather independent of the definition of `F-Race`: any reasonable sampling method may be considered.

13.4.1 Full Factorial Design

When `F-Race` was first proposed by Birattari et al. (2002), the candidate configurations were collected by a full factorial design (*FFD*) on the parameter space. The reason for adopting a full factorial design at that time was that it made more convenient the focus on the evaluation of `F-Race` and its comparison with other ways of defining races.

A full factorial design can be done by determining for each parameter a number of levels either manually, randomly or in some other way. Then, each possible combination of these levels represents a unique configuration, and Θ_0 comprises all possible combinations. One main drawback of a full factorial design is that it requires expertise to select the levels of each parameter. Maybe more importantly, the set of candidate configurations grows exponentially with the number of parameters. Suppose that d is the dimension of the parameter space and that each dimension has l levels; then the total number of candidate configurations would be l^d. It therefore quickly becomes impractical and computationally prohibitive to test all possible combinations, even for a reasonable number of levels at each dimension. We denote the version of `F-Race` using a full factorial design by `F-Race(FFD)`.

13.4.2 Random Sampling Design

The drawbacks of the full factorial design were described also by Balaprakash et al. (2007). They showed that `F-Race` with initial candidates generated by a random sampling design significantly outperforms the full factorial design for a number of applications. In the random sampling design, the initial elements are sampled according to some probability model p_X defined over the parameter space X.[7] If a priori information is available, such as the effects of certain parameters or their interactions, the probability model p_X can be defined accordingly. However, this is rarely the case, and the default way of defining the probability model p_X is to

[7] Note that the space of possible parameter value combinations X is different from the one-dimensional vector of candidate algorithm configurations Θ, and there exists a one-to-one mapping from X to Θ.

Algorithm 13.1: Iterated F-Race

```
Require : parameter space $X$, a noisy objective function black-box $f$.
initialize probability model $p_X$ for sampling from $X$;
set iteration counter $l = 1$;
repeat
    sample the initial set of configurations $\Theta_0^l$ based on $p_X$;
    evaluate set $\Theta_0^l$ by $f$ using F-Race;
    collect elite configurations from F-Race to update $p_X$;
    $l = l + 1$;
until termination criterion is met ;
identify the best parameter configuration $x^*$;
return $x^*$
```

assume a uniform distribution over X. We denote the random sampling version of F-Race based on uniform distributions by F-Race(RSD).

Two main advantages of the random sampling design are that of numerical parameters, no a priori definition of the levels needs to be done and that an arbitrary number of candidate configurations can be sampled while still covering the parameter space, on average, uniformly.

13.4.3 Iterated F-Race

As a next step, Balaprakash et al. (2007) proposed the iterative application of F-Race, where at each iteration a number of surviving candidate configurations of the previous iteration bias the sampling of new candidate configurations. It is hoped in this way to focus the sampling of candidate configurations around the most promising ones. In this sense, iterated F-Race follows directly the framework of model-based search (Zlochin et al. 2004), which is usually implemented in three steps: first, construct a candidate solution based on some probability model; second, evaluate all candidates; third, update the probability model of biasing the next sampling towards the better candidate solutions. These three steps are iterated, until some termination criterion is satisfied.

Iterated F-Race proceeds in a number of iterations. In each iteration, first a set of candidate configurations is sampled; this is followed by one run of F-Race applied to the sampled candidate configurations. An outline of the general framework of iterated F-Race is given in Alg. 13.1.

There are many possible ways in which iterated F-Race can be implemented. In fact, one possibility would be to use some algorithms for black-box mixed discrete–continuous optimization problems. However, a difficulty here may be that, for F-Race to be effective, the number of candidate configurations should be reasonably large, while due to the necessarily strongly limited number of function evaluations, few iterations should be run. Therefore, Balaprakash et al. (2007) followed a different approach and an ad hoc method was proposed for biasing the sampling. Unfortunately, there the ad hoc iterated `F-Race` was only defined and tested on numerical parameters. Nevertheless, it is relatively straightforward to generalize the ideas presented there to categorical parameters. In what follows, we first give a general discussion of the issues that arise in the definition of an iterated `F-Race` algorithm and then we present one particular implementation in Sect. 13.4.4. For the following discussion, we assume that the total computational budget B for the configuration process, which is measured by the number of function evaluations, is given a priori.

How many iterations? Iterated `F-Race` is an iterative process and therefore one needs to define the number of iterations. For a given computational budget, using few iterations will allow to sample at each iteration more candidate configurations and, hence, lead to more exploration at the cost of less possibilities of refining the model. In the extreme case of using only one iteration, this amounts to an execution of `F-Race(RSD)`. Intuitively, the number of iterations should depend on the number of parameters: if only few parameters are present, we expect, others things being equal, the problem to be less difficult to optimize and, hence, fewer iterations to be required.

Which computational budget at each iteration? Another issue concerns the distribution of the computational budget B among the iterations. The simplest idea is to divide the computational budget equally among all iterations. However, other possibilities are certainly reasonable; for example, one may decrease the number of function evaluations available with an increase of the iteration counter to increase exploration in the first iterations.

How many candidate configurations at each iteration? For `F-Race`, the number of candidate configurations to be sampled needs to be defined. A good idea is to make the number of candidate configurations dependent on the status of the race, in other words, the iteration counter. Typically, in the first iteration(s), the sampled candidate configurations are very different from each other, resulting in large performance differences. As a side-effect, poor candidate configurations usually can be quickly eliminated. In later iterations, the sampled candidate configurations become more similar and it becomes more difficult to determine the winner, that is, more instances are needed to detect significant differences among the configurations. Hence, for a same budget of function evaluations for one application of `F-Race`, in early iterations more configurations can be sampled, while in later iterations fewer candidate configurations should be generated to identify with a low variance a winning configuration.

When to terminate `F-Race` at each iteration? At each iteration l, `F-Race` terminates if one of the following two conditions is satisfied: (i) if the computational budget for the lth iteration, B_l, is spent; (ii) when a minimum number of candidate configurations, denoted by $N_{\min}$, remains. Another question concerns the value of $N_{\min}$. `F-Race` terminates by default if a unique survivor is identified. However, to maintain sufficient exploration of the parameter space, in iterated `F-Race` it may be better to keep a number of survivors at each iteration and to sample around these survivors the candidate configurations for the next iteration. Additionally, for setting $N_{\min}$, it may be a good idea to take into account the number of dimensions in the parameter space X: the larger the parameter space, the more survivors should remain to ensure sufficient exploration.

How should the candidate configurations be generated? As said, all candidate configurations are randomly sampled in the parameter space according to some probability distribution. For continuous and quasicontinuous parameters, continuous probability distributions are appropriate; for categorical and ordinal parameters, however, discrete probability distributions will be more useful. A first question related to the probability distributions is of which type they should be. For example, in the first paper on iterated F-Race (Balaprakash et al. 2007), normal distributions were chosen as models, but this choice need not be optimal. Another question related to the probability distributions is how they should be updated and, especially, how strong the bias towards the surviving configurations of the current iteration should be. Again, here the trade-off between exploration and exploitation needs to be taken into account.

13.4.4 An Example Iterated F-Race Algorithm

Here we describe one example implementation of iterated F-Race, which we refer to as I/F-Race in the following. This example implementation is based on the previous one published by Balaprakash et al. (2007). However, it differs in some parameter choices and extends the earlier version by defining a way to handle categorical parameters. Note that the proposed parameter settings are chosen in an ad hoc version; tuning the parameter settings of I/F-Race is beyond the scope of this chapter.

Number of iterations. We denote by L the *number of iterations* of I/F-Race, and increase L with d, the number of parameters, using a setting of $L = 2 + \mathrm{round}(\log_2 d)$.

Computational budget at each iteration. The *computational budget* is distributed as equally as possible across the iterations. B_l, the computational budget in iteration l, where $l = 1, ..., L$, is set to $B_l = (B - B_{\mathrm{used}})/(L - l + 1)$; B_{used} denotes the total computational budget used until iteration $l - 1$.

The number of candidate configurations. We introduce a parameter μ_l, and set the number of candidate configurations sampled at iteration l to be $N_l = \lfloor B_l/\mu_l \rfloor$. We let μ_l increase with the number of iterations, using a setting of $\mu_l = 5 + l$. This allows more evaluation steps to identify the winners when the configurations are deemed to become more similar.

Termination of F-Race at each iteration. In addition to the usual termination criteria of F-Race, we stop it if at most $N_{\min} = 2 + \mathrm{round}(\log_2 d)$ candidate configurations remain.

Generation of candidate configurations. In the first iteration, all candidate configurations are sampled uniformly at random. Once F-Race terminates, the best N_s candidate configurations are selected for the update of the probability model.

We use $N_s = \min(N_{\text{survive}}, N_{\min})$, where N_{survive} denotes the number of candidates that survive the race. These N_s elite configurations are then weighted according to their ranks, where the weight of an elite configuration with rank r_z ($z = 1, \ldots, N_s$) is given by

$$w_z = \frac{N_s - r_z + 1}{N_s \cdot (N_s + 1)/2}. \tag{13.3}$$

In other words, the weight of an elite configuration is inversely proportional to its rank. Since the instances for configuration are sampled randomly from the training set, the N_s elite configurations of the lth iteration will be re-evaluated in the $(l+1)$st iteration, together with the $N_{l+1} - N_s$ candidate configurations to be sampled anew. (Alternatively, it is possible to evaluate the configurations on fixed instances, so that the results of the elite configurations from the last iteration could be reused.) The $N_{l+1} - N_s$ new candidate configurations are iteratively sampled around one of the elite configurations. To do so, for sampling each new candidate configuration, first one elite solution E^z ($z \in \{1, \ldots, N_s\}$) is chosen with a probability proportional to its weight w_z and next a value is sampled for each parameter. The sampling distribution of each parameter depends on whether it is a numerical one (the set of such parameters being denoted by X^{num}) or a categorical one (the set of such parameters being denoted by X^{cat}). We have that the parameter space $X = X^{\text{num}} \cup X^{\text{cat}}$.
First suppose that X_i is a numerical parameter, i.e. $X_i \in X^{num}$, with boundary $X_i \in [\underline{X_i}, \overline{X_i}]$. Denote by $v_i = \overline{X_i} - \underline{X_i}$ the range of the parameter X_i. The sampling distribution of X_i follows a normal distribution $N(x_i^z, \sigma_i^l)$, with x_i^z being the mean and σ_i^l being the standard deviation of X_i in the lth iteration. The standard deviation is reduced in a geometric fashion from iteration to iteration using a setting of

$$\sigma_i^{l+1} = v_i \cdot \left(\frac{1}{N_{l+1}}\right)^{\frac{l}{d}} \qquad \text{for } l = 1, \ldots, L-1. \tag{13.4}$$

In other words, the standard deviation for the normal distribution is reduced by a factor of $\left(\frac{1}{N_{l+1}}\right)^{\frac{1}{d}}$ as the iteration counter increments. Hence, the more parameters, the smaller the update factor becomes, resulting in a stronger bias of the elite configuration on the sampling. Furthermore, the larger the number of candidate configurations to be sampled, the stronger the bias of the sampling distribution.
Now, suppose that $X_i \in X^{\text{cat}}$ with n_i levels $F_i = f_1, \ldots, f_{n_i}$. Then we use a discrete probability distribution $p_l(F_i)$ with iteration $l = 1, \ldots, L$, and initialize p_1 to be uniformly distributed over F_i. Suppose further that after the lth iteration ($l > 1$), the ith parameter of the selected elite configuration E^z takes level f_i^z. Then, the discrete distribution of parameter X_i is updated as

$$p_{l+1}(f_j) = p_l(f_j)\cdot(1-\frac{l}{L})+I_{j=f_i^z}\cdot\frac{l}{L} \quad \text{for } l = 1,\dots,L-1 \text{ and } j = 1,\dots,n_i \tag{13.5}$$

where I is an indicator function; the bias of the elite configuration on the sampling distribution is getting stronger as the iteration counter increments.
The conditional parameters are sampled only when they are activated by their associated upper-level categorical parameter, and their sampling model is updated only when they appear in elite configurations.

13.5 Case Studies

In this section, we experimentally evaluate the presented variant of `I/F-Race` and we compare it in three case studies to `F-Race(RSD)` and `F-Race(FFD)`.

All three case studies concern the configuration of ant colony optimization (ACO) algorithms applied to the traveling salesman problem (TSP). They are ordered according to the number of parameters to be tuned. In particular, they involve configuring $\mathcal{MAX}$–$\mathcal{MIN}$ *Ant System* ($\mathcal{MM}$AS), a particularly successful ACO algorithm (Stützle and Hoos 2000), using four categorical parameters and configuring $\mathcal{MM}$AS using seven categorical parameters. Both case studies use the $\mathcal{MM}$AS implementation available in the ACOTSP software package.[8] The ACOTSP package implements several ACO algorithms for the TSP. The third case study uses the ACOTSP package as a black-box software and involves setting 12 mixed parameters. Among others, one of these parameters is the choice of which ACO algorithm should be used.

In all experiments we used Euclidean TSP instances with 750 nodes, where the nodes are uniformly distributed in a square of side length $10,000$. We generated $1,000$ instances for training and 300 for evaluating the winning configurations using the DIMACS instance generator (Johnson et al. 2001). The experiments were carried out on cluster computing nodes, each equipped with two quad-core XEON E5410 CPUs running at 2.33 GHz with 2×6 MB second-level cache and 8 GB RAM. The cluster was running under Cluster Rocks Linux version 4.2.1/CentOS 4. The programme was compiled with gcc-3.4.6-3, and only one CPU core was used for each run due to the sequential implementation of the ACOTSP software.

For each case study we have run a total of six experiments, which result from all six combinations of two different computation time limits allocated for each `function evaluation` to the ACOTSP software (5 and 20 CPU seconds) and three values for the computational budget. The different levels of the computational budget have been chosen to examine the dependence of the possible advantage of `I/F-Race` as a function of the corresponding computational budget.

In each of the six experiments, ten `trials` were run. Each `trial` is the execution of the `configuration process`, in our case, either `F-Race(FFD)`,

[8] The ACOTSP package is available at `http://www.aco-metaheuristic.org/aco-code/`.

Parameter	Range	No. of levels
α	$[0.01, 5.00]$	11
β	$[0.01, 10.00]$	11
ρ	$[0.00, 1.00]$	10
m	$[5, 100]$	10

Table 13.1: The parameters, the original range considered before discretization, and the number of levels considered after discretization for the first case study. The number of candidate parameter settings is $12,100$

`F-Race(RSD)`, or `I/F-Race`, together with a subsequent `testing procedure`. In the testing procedure, the final parameter setting returned by configuration process is evaluated on 300 test instances.

13.5.1 Case Study 1: $\mathcal{MM}$AS under Four Parameters

In this case study, we tune four parameters of $\mathcal{MM}$AS: the relative influence of pheromone trails α, the relative influence of heuristic information β, the pheromone evaporation rate ρ, and the number of ants m.

In this first and the second case study, we discretize these numerical parameters and treat them as categorical ones. Each parameter is discretized by regular grids, resulting in a relatively large number of levels. Their ranges and number of levels as listed in Table 13.1.[9] The motivation for discretizing numerical parameters is to test whether `I/F-Race` is able to improve over `F-Race(RSD)` and `F-Race(FFD)` for categorical parameters; previously, it was already shown that `I/F-Race` gives advantages for numerical parameters (Balaprakash et al. 2007).

The three levels of the computational budget chosen are $6 \cdot 3^4 = 486$, $6 \cdot 4^4 = 1,536$, and $6 \cdot 5^4 = 3,750$. In this way the candidate generation of `F-Race(FFD)` can be done by selecting the same number of levels for each parameter, in our case three, four, and five. Without a priori knowledge, the level of each parameter is selected randomly in `F-Race(FFD)`.

The experimental results are given in Table 13.2. The table shows the average percentage deviation of each algorithm from the reference cost, which for each instance is defined by the average cost across all candidate algorithms on that instance. The results of the algorithms tuned by `F-Race(FFD)`, `F-Race(RSD)`, and `I/F-Race`, are compared using the nonparametric pairwise Wilcoxon test with Holm adjustment, using blocking on the instances; the significance level chosen is 0.05. Results in boldface indicate that the corresponding configurations are statistically better than the ones of the two competitors.

[9] For the other parameters, we use default values and we opted for an ACO version that does not use local search.

	5 seconds	20 seconds	
algo	per.dev	per.dev	max.bud
F-Race(FFD)	+0.85	+0.79	486
F-Race(RSD)	**−0.58**	−0.44	486
I/F-Race	−0.26	−0.34	486
F-Race(FFD)	+0.51	+1.27	1 536
F-Race(RSD)	−0.08	−0.66	1 536
I/F-Race	**−0.42**	−0.61	1 536
F-Race(FFD)	+0.40	+0.71	3 750
F-Race(RSD)	−0.12	−0.27	3 750
I/F-Race	**−0.28**	**−0.45**	3 750

Table 13.2: Computational results for configuring $\mathcal{MM}$AS for the TSP with four discretized parameters for a computation time bound of 5 and 20 s, respectively. The column entries with the label `per.dev` show the mean percentage deviation of each algorithm from the reference cost. $+x$ ($-x$) means that the solution cost of the algorithm is $x\%$ more (less) than the reference cost. The column with the label `max.bud` gives the maximum number of evaluations given to each algorithm

In all experiments, `I/F-Race` and `F-Race(RSD)` significantly outperform `F-Race(FFD)`. Overall, `I/F-Race` has a slight advantage over `F-Race(RSD)`: in three of six experiments `I/F-Race` returns configurations that are significantly better than those found by `F-Race(RSD)`, while the opposite is true in only one experiment. The trend appears to be that, with larger total budget, the advantage of `I/F-Race` over `F-Race(RSD)` increases. The reason for the relatively good performance of `F-Race(RSD)` could be due to the fact that the parameter space is rather small (12,100 candidate configurations) and that the number of levels (10 or 11) for each parameter is large.

13.5.2 Case Study 2: $\mathcal{MM}$AS under Seven Parameters

In this case study we have chosen seven parameters. These are the same as in the first case study plus three additional parameters: γ, a parameter that controls the gap between the minimum and maximum pheromone trail value in $\mathcal{MM}$AS, $\gamma = \tau_{\max}/(\tau_{\min} \cdot instance\_size)$; nn, the number of nearest neighbors used in the solution construction phase; and q_0, the probability of selecting the best neighbor deterministically in the pseudorandom proportional action choice rule; for a detailed definition see Dorigo and Stützle (2004).

The parameters are discretized using the ranges and number of levels given in Table 13.3. Note that, in comparison with the previous experiment, the parameter space is more than one order of magnitude larger ($259,200 \gg 12,100$). Besides, there is a smaller number of levels for each parameter, usually between four to nine. We use the same experimental setup as in the previous section, except that for the computational budget, we choose $6 \cdot 2^7 = 768$ such that each parameter in

Parameter	Range	No. of levels
α	$[0.01, 5.00]$	5
β	$[0.01, 10.00]$	6
ρ	$[0.00, 1.00]$	8
γ	$[0.01, 5.00]$	6
m	$[5, 100]$	5
nn	$[5, 50]$	4
q_0	$[0.0, 1.0]$	9

Table 13.3: The parameters, the original range considered before discretization, and the number of levels considered after discretization for the first case study. The number of candidate parameter settings is $259, 200$

	5 seconds	20 seconds	
algo	per.dev	per.dev	max.bud
F-Race(FFD)	+9.33	+4.61	768
F-Race(RSD)	−4.49	−1.35	768
I/F-Race	**−4.84**	**−3.25**	768
F-Race(FFD)	+1.58	+2.11	1 728
F-Race(RSD)	−0.49	−0.78	1 728
I/F-Race	**−1.10**	**−1.33**	1 728
F-Race(FFD)	+0.90	+2.38	3 888
F-Race(RSD)	−0.27	−0.33	3 888
I/F-Race	**−0.63**	**−2.05**	3 888

Table 13.4: Computational results for configuring $\mathcal{MM}$AS for TSP with seven categorical parameters in 5 and 20 CPU s. For an explanation of the table entries see the caption of Table 13.2

F-Race(FFD) has two levels, $6 \cdot 2^5 \cdot 3^2 = 1\,728$, such that in F-Race(FFD), five parameters will have two levels and the other two three levels, and $6 \cdot 2^3 \cdot 3^4 = 3,888$, such that in F-Race(FFD), three parameters will have two levels, and the other four parameters have three levels.

The experimental results are listed in Table 13.4 and the results are analyzed in a way analogous to case study 1. The results clearly show that I/F-Race significantly outperforms F-Race(FFD) and F-Race(RSD) in each experiment. As expected, also F-Race(RSD) outperforms F-Race(FFD) significantly.

13.5.3 Case Study 3: ACOTSP under 12 Parameters

In a final experiment, 12 parameters of the ACOTSP software are examined. This configuration task is the most complex and it requires the setting of categorical as well as numerical parameters.

Among these parameters, firstly two categorical parameters have to be determined: (i) the choice of the ACO algorithm, among the five variants $\mathcal{MM}$AS, ant colony system (ACS), rank-based ant system (RAS), elitist ant system (EAS), and

	5 seconds	20 seconds	
algo	per.dev	per.dev	max.bud
F-Race(RSD)	+0.06	+0.005	1 500
I/F-Race	**−0.06**	**−0.005**	1 500
F-Race(RSD)	+0.04	+0.009	3 000
I/F-Race	**−0.04**	**−0.009**	3 000
F-Race(RSD)	+0.07	−0.001	6 000
I/F-Race	**−0.07**	+0.001	6 000

Table 13.5: Computational results for configuring $\mathcal{MM}$AS for TSP with 12 parameters in 5 and 20 CPU s. For an explanation of the table entries see the caption of Table 13.2

ant system (AS); (ii) the local search type l, including four levels: no local search, 2-opt, 2.5-opt, and 3-opt. All the ACO construction algorithms share the three continuous parameter α, β, and ρ, and two quasicontinuous parameters m and nn, which have been introduced before. Moreover, five conditional parameters are considered: (i) the continuous parameter q_0 (introduced in Sect. 13.5.2) is only in effect when ACS is deployed; (ii) the quasi-continuous $rasrank$ is only in effect when RAS is chosen; (iii) the quasi-continuous $eants$ is only in effect when EAS is applied; (iv) the quasi-continuous parameter $nnls$ is only in effect when local search is used, namely either 2-opt, 2.5-opt or 3-opt; (v) the categorical parameter dlb is only in effect when local search is used. Here, only F-Race(RSD) and I/F-Race are tested because F-Race(FFD) has so far already been outperformed by the other two variants, and due to the large number of parameters, running F-Race(FFD) becomes infeasible. As computational budgets we adopted $1,500$, $3,000$, and $6,000$ function evaluations. The experimental results are given in Table 13.5. The two algorithms F-Race(RSD) and I/F-Race are compared using the nonparametric pairwise Wilcoxon test using a 0.05 significance level. The statistical comparisons show that I/F-Race is again dominating. It is significantly better performing in five out of six experiments; only in one case can no statistically significant difference be identified. However, the quality differences in this set of experiments are quite small, usually below 0.1% in the 5 CPU seconds case, while in the 20 CPU seconds case the difference is below 0.01%. This shows that the solution quality is not very sensitive to the parameter settings. This is usually the case when a strong local search such as 3-opt is used.

13.6 A Review of F-Race Applications

F-Race has received significant attention, as witnessed by the 99 citations to the first article on F-Race (Birattari et al. 2002) in the Google Scholar database as of June 2009. In what follows, we give an overview of research that applied F-Race in various contexts.

Fine-tuning algorithms The by far most common use of `F-Race` is to use it as a method to fine-tune an existing or a recently developed algorithm. Often, tuning through `F-Race` is also done before comparing the performance of various algorithms. In fact, this latter usage is important to make reasonably sure that performance differences between algorithms are not simply due to uneven tuning.

A significant fraction of the usages of `F-Race` is due to researchers either involved in the development of the `F-Race` method or by their collaborators. In fact, `F-Race` has been developed in the research for the Metaheuristics Network, an EU-funded research and training network on the study of metaheuristics. Various applications there have been for configuring different metaheuristics for the university-course timetabling problem (Chiarandini and Stützle 2002, Manfrin 2003, Rossi-Doria et al. 2003) and also for various other problems (Chiarandini 2005, Chiarandini and Stützle 2007, den Besten 2004, Risler et al. 2004, Schiavinotto and Stützle 2004).

Soon after these initial applications, `F-Race` was also adopted by a number of other researchers. Most applications focus on configuring SLS methods for combinatorial optimization problems (Bin Hussin et al. 2007, Balaprakash et al. 2009a, Di Gaspero and Roli 2008, Di Gaspero et al. 2007, Lenne et al. 2007, Pellegrini 2005, Philemotte and Bersini 2008). However, also other applications have been considered, including the tuning of algorithms for training neural networks (Blum and Socha 2005, Socha and Blum 2007) or the tuning of parameters of a control system for simple robots (Nouyan 2008, Nouyan et al. 2008).

Industrial applications Few researches have evaluated `F-Race` in pilot studies for industrial applications. The first has been a feasibility study, where `F-Race` was used to configure a commercial solver for vehicle routing and scheduling problems, which has been developed by the software company SAP. In this research, six configuration tasks have been considered that ranged from the study of specific parameters, which determined the frequency of the application of some important operators of the program, to the configuration of the SLS method that was used in the software package. `F-Race` was compared with a strategy that after each fixed number of instances discarded a fixed percentage of the worst candidate configurations, showing, as expected, advantages for `F-Race` when the performance differences between configurations were stronger. Some results of this study have been published by Becker et al. (2005); more details are available in a master thesis (Becker 2004).

Yuan et al. (2008) have adopted `F-Race` to configure several algorithms for a highly constrained train scheduling problem arising at Deutsche Bahn AG. A comparison of various tuned algorithms identified an iterated greedy algorithm as the most promising one.

Algorithm development `F-Race` has occasionally also been used to explicitly support the algorithm development process. A first example is described by Chiarandini et al. (2006) who used `F-Race` to design a hybrid metaheuristic for the university-course timetabling problem. In their work they have adopted `F-Race` in a semi-automatic way. They observed the algorithm candidates that were maintained in a race and based on this information they generated new algorithm candidates that were then manually added to the ongoing race. In fact, one of these newly injected candidate algorithms was finally the best performing algorithm in an international timetabling competition (see also `http://www.idsia.ch/Files/ttcomp2002`).

The PhD work of den Besten (2004) provides an empirical investigation into the application of ILS to solve a range of deterministic scheduling problems with tardiness penalties. Racing in general, and F-Race in particular, is a very important ingredient throughout the algorithm development and calibration. The ILS algorithms are built in a modular way and F-Race is applied to assess each combination of modular components of the algorithm.

Comparison of F-Race with other methods There have been some comparisons of F-Race with other racing algorithms. Some preliminary results comparing F-Race and *t*-test-based racing techniques are presented by Birattari (2004b, 2009), showing that F-Race typically performs best.

Yuan and Gallagher (2004) discuss the use of F-Race for the empirical evaluation of evolutionary algorithms. They also use an algorithm called *A-Race*, where the family-wise test is based on the *analysis of variance* (ANOVA) method. From the experiments they conduct, they conclude that their version of F-Race obtains better results than *A-Race*.

In their work Caelen and Bontempi (2005) compare five techniques from various communities on a model selection task. The techniques compared are (i) a two-stage selection technique proposed in the stochastic simulation community, (ii) a stochastic dynamic programming approach conceived to address the multi-armed bandit problem, (iii) a racing method, (iv) a greedy approach, and (v) a round-search technique. F-Race is mentioned and applied for comparison purposes. The comparison results shows that the bandit strategy yields the most promising performance when the sample size is small, but F-Race outperforms other techniques when the sample size is sufficiently large.

Extensions and hybrids of F-Race The F-Race algorithm has been adopted as a module integrated into an ACO algorithm framework for tackling combinatorial optimization problems under uncertainty (Birattari et al. 2007). The resulting algorithm is called ACO/F-Race and it uses F-Race to determine the best of a set of candidate solutions generated by the ACO algorithm. In later work by Balaprakash et al. (2009b) on the application of estimation-based ACO algorithms to the probabilistic traveling salesman problem the Friedman test is replaced by an ANOVA.

Yuan and Gallagher (2005, 2007) propose an approach to tune evolutionary algorithms by hybridizing Meta-EA and F-Race. Meta-EA is an approach that uses various genetic operators to tune the parameters of EAs. It is reported that one major difficulty in Meta-EA is that it cannot effectively handle categorical parameters. These categorical parameters are usually handled in Meta-EA by pure random search. The proposed hybridization uses Meta-EA to evolve part of the numerical parameters and leave the categorical parameters for F-Race. Experiment show that

Meta-EA plus `F-Race` required only around 12% of the computational effort taken by Meta-EA plus random search.

13.7 Summary and Outlook

In this chapter, we have presented the algorithm configuration problem that `F-Race` tackles and have given a detailed review of the method. `F-Race` is essentially a method for selecting the best algorithm configuration under stochastic evaluations. As such, it is a method that is independent of the way the candidate configurations are sampled. In a next step, we have introduced the family of iterated `F-Race` algorithms, where the sampling of new candidate configurations is done through probability models that are iteratively refined.

There is a significant number of possible extensions and adaptations of the `F-Race` method. In fact, any mixed-integer (stochastic) optimization techniques could, at least in principle, provide the sampling method for iterated `F-Race`. A part of our current research is actually devoted to this observation. We are currently studying the usage of `F-Race` on top of continuous optimization methods, and first results show statistically significant advantages over strategies using a fixed sample size. Combinations of `F-Race` with other methods for parameter tuning such as SPO (Bartz-Beielstein 2006) and local search approaches (Hutter et al. 2007) may be also useful. Finally, we believe that the ideas on which `F-Race` is based can be also fruitful for tasks other than algorithm tuning. In fact, we envision that especially applications to stochastic optimization problems may benefit greatly, `ACO/F-Race` (Birattari et al. 2007) being a first such successful example.

Acknowledgements This work has been supported by META-X, an ARC project funded by the French Community of Belgium. The authors acknowledge support from the fund for scientific research FRS-FNRS of the French Community of Belgium, of which they are research associates (M.B. and T.S.), or aspirant (Z.Y.), respectively.

References

Adenso-Diaz B, Laguna M (2006) Fine-tuning of algorithms using fractional experimental designs and local search. Operations Research 54(1):99–114

Balaprakash P, Birattari M, Stützle T (2007) Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. In: Bartz-Beielstein T, et al. (eds) Hybrid Metaheuristics, 4th International Workshop, HM 2007, Lecture Notes in Computer Science, vol 4771, Springer, Berlin, Germany, pp. 108–122

Balaprakash P, Birattari M, Stützle T, Dorigo M (2009a) Adaptive sample size and importance sampling in estimation-based local search for the probabilistic traveling salesman problem. European Journal of Operational Research 199(1):98–110

Balaprakash P, Birattari M, Stützle T, Yuan Z, Dorigo M (2009b) Ant colony optimization and estimation-based local search for the probabilistic traveling salesman problem. Swarm Intelligence 3(3):223–242

Bartz-Beielstein T (2006) Experimental Research in Evolutionary Computation. Springer, Berlin, Germany

Becker S (2004) Racing-Verfahren für Tourenplanungsprobleme. Diplomarbeit, Technische Universität Darmstadt, Darmstadt, Germany

Becker S, Gottlieb J, Stützle T (2005) Applications of racing algorithms: An industrial perspective. In: Talbi EG, et al. (eds) Artificial Evolution: 7th International Conference, Evolution Artificielle, EA 2005, Springer Verlag, Berlin, Germany, Lille, France, Lecture Notes in Computer Science, vol 3871, pp. 271–283

den Besten ML (2004) Simple metaheuristics for scheduling. an empirical investigation into the application of iterated local search to deterministic scheduling problems with tardiness penalities. PhD thesis, FG Intellektik, FB Informatik, TU Darmstadt

Billingsley P (1986) Probability and Measure, 2nd edn. Wiley, New York, NY, USA

Bin Hussin MS, Stützle T, Birattari M (2007) A study of stochastic local search algorithms for the quadratic assignment problems. In: Ridge E, et al. (eds) Proceedings of SLS-DS 2007, Doctoral Symposium on Engineering Stochastic Local Search Algorithms, Brussels, Belgium, pp. 11–15

Birattari M (2004a) On the estimation of the expected performance of a metaheuristic on a class of instances. How many instances, how many runs? Tech. Rep. TR/IRIDIA/2004-001, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium

Birattari M (2004b) The problem of tuning metaheuristics as seen from a machine learning perspective. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium

Birattari M (2009) Tuning Metaheuristics: A Machine Learning Perspective, Studies in Computational Intelligence, vol 197. Springer, Berlin, Germany

Birattari M, Stützle T, Paquete L, Varrentrapp K (2002) A racing algorithm for configuring metaheuristics. In: Langdon WB, et al. (eds) GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann Publishers, San Francisco, CA, pp. 11–18

Birattari M, Balaprakash P, Dorigo M (2007) The ACO/F-Race algorithm for combinatorial optimization under uncertainty. In: Doerner KF, et al. (eds) Metaheuristics - Progress in Complex Systems Optimization, Operations Research/Computer Science Interfaces Series, Springer, Berlin, Germany, pp. 189–203

Blum C, Socha K (2005) Training feed-forward neural networks with ant colony optimization: An application to pattern classification. In: Nedjah N, et al. (eds) Proceedings of Fifth International Conference on Hybrid Intelligent Systems (HIS'05), IEEE Computer Society, Los Alamitos, CA, USA, pp. 233–238

Burnham K, Anderson D (2002) Model selection and multimodel inference: a practical information-theoretic approach. Springer

Caelen O, Bontempi G (2005) How to allocate a restricted budget of leave-one-out assessments for effective model selection in machine learning: a comparison of state-of-the-art techniques. In: Verbeeck K, et al. (eds) Proceedings of the 17th Belgian-Dutch Conference on Artificial Intelligence (BNAIC'05), Brussels, Belgium, pp. 51–58

Chiarandini M (2005) Stochastic local search methods for highly constrained combinatorial optimisation problems. PhD thesis, Technische Universität Darmstadt, Darmstadt, Germany

Chiarandini M, Stützle T (2002) Experimental evaluation of course timetabling algorithms. Tech. Rep. AIDA-02-05, FG Intellektik, FB Informatik, Technische Universität Darmstadt, Darmstadt, Germany

Chiarandini M, Stützle T (2007) Stochastic local search algorithms for graph set t-colouring and frequency assignment. Constraints 12(3):371–403

Chiarandini M, Birattari M, Socha K, Rossi-Doria O (2006) An effective hybrid algorithm for university course timetabling. Journal of Scheduling 9(5):403–432

Conover WJ (1999) Practical Nonparametric Statistics, 3rd edn. Wiley, New York, NY, USA

Dean A, Voss D (1999) Design and Analysis of Experiments. Springer, New York, NY, USA

Di Gaspero L, Roli A (2008) Stochastic local search for large-scale instances of the haplotype inference problem by pure parsimony. Journal of Algorithms 63(1-3):55–69

Di Gaspero L, di Tollo G, Roli A, Schaerf A (2007) Hybrid local search for constrained financial portfolio selection problems. In: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Lecture Notes in Computer Science, vol 4510, Springer Verlag, Berlin, Germany, pp. 44–58

Dorigo M, Stützle T (2004) Ant colony optimization. MIT Press, Cambridge, MA

Hoos HH, Stützle T (2004) Stochastic Local Search. Foundations and Applications. Morgan Kaufmann, San Francisco, CA, USA

Hutter F, Hoos HH, Stützle T (2007) Automatic algorithm configuration based on local search. In: Holte RC, et al. (eds) Proceedings of the 22nd Conference on Artificial Intelligence (AAAI), AAAI Press / The MIT Press, Menlo Park, CA, USA, pp. 1152–1157

Johnson DS, McGeoch LA, Rego C, Glover F (2001) 8th DIMACS implementation challenge. `http://www.research.att.com/~dsj/chtsp/` (webpage last visited in April 2009)

Lenne R, Solnon C, Stützle T, Tannier E, Birattari M (2007) Effective stochastic local search algorithms for the genomic median problem. In: Ridge E, et al. (eds) Proceedings of SLS-DS 2007, Doctoral Symposium on Engineering Stochastic Local Search Algorithms, Brussels, Belgium, pp. 1–5

Manfrin M (2003) Metaeuristiche per la costruzione degli orari dei corsi universitari. Tesi di Laurea, Università degli Studi di Firenze, Firenze, Italy, in Italian

Maron O (1994) Hoeffding races: Model selection for MRI classification. Master's thesis, The Massachusetts Institute of Technology, Cambridge, MA, USA

Maron O, Moore AW (1994) Hoeffding races: Accelerating model selection search for classification and function approximation. In: Cowan JD, et al. (eds) Advances in Neural Information Processing Systems, Morgan Kaufmann, San Francisco, CA, USA, vol 6, pp. 59–66

Maron O, Moore AW (1997) The racing algorithm: Model selection for lazy learners. Artificial Intelligence Review 11(1–5):193–225

Montgomery DC (2000) Design and Analysis of Experiments, 5th edn. Wiley, New York, NY, USA

Nouyan S (2008) Teamwork in a swarm of robots – an experiment in search and retrieval. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium

Nouyan S, Campo A, Dorigo M (2008) Path formation in a robot swarm. Swarm Intelligence 2(1):1–23

Papoulis A (1991) Probability, Random Variables, and Stochastic Processes, 3rd edn. McGraw-Hill, New York, NY, USA

Pellegrini P (2005) Application of two nearest neighbor approaches to a rich vehicle routing problem. Tech. Rep. TR/IRIDIA/2005-15, IRIDIA, Université Libre de Bruxelles, Belgium

Philemotte C, Bersini H (2008) The gestalt heuristic: learning the right level of abstraction to better search the optima. Tech. Rep. TR/IRIDIA/2008-021, IRIDIA, Université Libre de Bruxelles, Belgium

Risler M, Chiarandini M, Paquete L, Schiavinotto T, Stützle T (2004) An algorithm for the car sequencing problem of the ROADEF 2005 challenge. Tech. Rep. AIDA–04–06, FG Intellektik, TU Darmstadt, Darmstadt, Germany

Rossi-Doria O, Sampels M, Birattari M, Chiarandini M, Dorigo M, Gambardella LM, Knowles J, Manfrin M, Mastrolilli M, Paechter B, Paquete L, Stützle T (2003) A comparison of the performance of different metaheuristics on the timetabling problem. In: Burke E, et al. (eds) Practice and Theory of Automated Timetabling IV, Springer Verlag, Berlin, Germany, Lecture Notes in Computer Science, vol 2740, pp. 329–351

Schiavinotto T, Stützle T (2004) The linear ordering problem: Instances, search space analysis and algorithms. Journal of Mathematical Modelling and Algorithms 3(4):367–402

Sheskin D (2000) Handbook of Parametric and Nonparametric Statistical Procedures, 2nd edn. Chapman & Hall/CRC, Boca Raton, FL, USA

Siegel S, Castellan NJ Jr (1988) Nonparametric Statistics for the Behavioral Sciences, 2nd edn. McGraw-Hill, New York, NY, USA

Socha K, Blum C (2007) An ant colony optimization algorithm for continuous optimization: application to feed-forward neural network training. Neural Computing and Applications 16(3):235–247

Stützle T, Hoos HH (2000) $\mathcal{MAX}$–$\mathcal{MIN}$ ant system. Future Generation Computer Systems 16(8):889–914

Yuan B, Gallagher M (2004) Statistical racing techniques for improved empirical evaluation of evolutionary algorithms. In: Yao X, et al. (eds) Parallel Problem Solving from Nature - PPSN VIII, Lecture Notes in Computer Science, vol 3242, Springer Verlag, Berlin, Germany, pp. 172–181

Yuan B, Gallagher M (2005) A hybrid approach to parameter tuning in genetic algorithms. In: Proceedings of the IEEE Congress in Evolutionary Computation (CEC'05), IEEE Press, Piscataway, NJ, vol 2, pp. 1096–1103

Yuan B, Gallagher M (2007) Combining Meta-EAs and racing for difficult EA parameter tuning tasks. In: Parameter Setting in Evolutionary Algorithms, Studies in Computational Intelligence, vol 54, Springer Verlag, Berlin, Germany, pp. 121–142

Yuan Z, Fügenschuh A, Homfeld H, Balaprakash P, Stützle T, Schoch M (2008) Iterated greedy algorithms for a real-world cyclic train scheduling problem. In: Blesa MJ, et al. (eds) Hybrid Metaheuristics, 5th International Workshop, HM 2008, Springer Verlag, Berlin, Germany, Lecture Notes in Computer Science, vol 5296, pp. 102–116

Zlochin M, Birattari M, Meuleau N, Dorigo M (2004) Model-based search for combinatorial optimization: A critical survey. Annals of Operations Research 131(1–4):373–395

Chapter 14
The Sequential Parameter Optimization Toolbox

Thomas Bartz-Beielstein, Christian Lasarczyk, and Mike Preuss

Abstract The *sequential parameter optimization toolbox* (SPOT) is one possible implementation of the SPO framework introduced in Chap. 2. It has been successfully applied to numerous heuristics for practical and theoretical optimization problems. We describe the mechanics and interfaces employed by SPOT to enable users to plug in their own algorithms. Furthermore, two case studies are presented to demonstrate how SPOT can be applied in practice, followed by a discussion of alternative metamodels to be plugged into it. We conclude with some general guidelines.

14.1 Introduction

The sequential parameter optimization approach discussed in Chap. 2 is a flexible and general framework which can be applied in many situations. Here, we introduce the *sequential parameter optimization toolbox* (SPOT) as one possible implementation of this framework. The basic ideas of this approach can be described as follows:

1. Use the available budget sequentially, i.e., use information from the exploration of the search space to guide the search by building one or several meta models. Choose new design points based on predictions from the meta model(s). Refine the meta model(s) stepwise to improve knowledge about the search space.

Thomas Bartz-Beielstein
Institute of Computer Science, Cologne University of Applied Sciences, 51643 Gummersbach, Germany, e-mail: thomas.bartz-beielstein@fh-koeln.de

Christian Lasarczyk
Algorithm Engineering, TU Dortmund, Germany, e-mail: Christian.Lasarczyk@udo.edu

Mike Preuss
Algorithm Engineering, TU Dortmund, Germany, e-mail: mike.preuss@tu-dortmund.de

2. Try to cope with noise by improving confidence. Guarantee comparable confidence for search points.
3. Collect information to learn from this tuning process, e.g., apply explorative data analysis (Tukey 1991).
4. Provide mechanisms both for interactive and automated tuning.

We consider SPOT as one of the most effective and efficient tuning procedures that enables learning from experiments. It was developed over recent years by Thomas Bartz-Beielstein, Christian Lasarczyk, and Mike Preuss (Bartz-Beielstein et al. 2005). The main purpose of SPOT is to determine improved parameter settings for optimization algorithms to analyze and understand their performance. SPOT was successfully applied to numerous optimization algorithms, especially in the field of evolutionary computation.

The remainder of this chapter is organized as follows. Section 14.2 presents some typical examples. Section 14.3 describes goals of the SPOT approach. In Sect. 14.4, elements of SPOT are explained. Since SPOT requires some statistical considerations, these will be discussed in Sect. 14.5. A case study that performs a regression analysis of the (1+1)-ES from Chap. 2 is presented in Sect. 14.6. Besides classical regression, SPOT allows the use of modern models such as tree-based regressions; pros and cons of these models are discussed in Sect. 14.7. The sequential approach is reconsidered in Sect. 14.8. Results from this chapter are summarized in Sect. 14.9.

14.2 Applications

This section presents typical applications of the sequential parameter optimization toolbox from bioinformatics, water-resource management, mechanical engineering, biogas plant simulation, shipbuilding, and quality control. Bartz-Beielstein (2010a) lists more than one hundred publications which mention how SPOT can be applied to evolution strategies, particle swarm optimization, genetic programming, and many more algorithms.

14.2.1 Bioinformatics

Volkert (2006) discusses results from a sequential parameter optimization approach for investigating the effectiveness of using EAs for training *hidden Markov models* (HMM). She concludes that "this approach not only offers the possibility for improving the effectiveness of the EA but will also provide much needed insight into directions for future improvements in the design of EAs for the construction of HMMs in general."

Fober et al. (2009) adjusted seven exogenous parameters of an EA using the sequential parameter optimization toolbox. The EA was used for a structural analysis of biomolecules.

14.2.2 Water-Resource Management

The aim of a paper by Konen et al. (2009) is the prediction of fill levels in stormwater tanks based on rain measurements and soil conditions. Different prediction methods are compared. The sequential parameter optimization is used to find in a comparable manner the best parameters for each method. Main results of this work are: (i) SPOT is applicable to diverse forecasting methods and automates the time-consuming parameter tuning, (ii) the best manual result achieved before was improved with SPOT by 30% and (iii) SPOT analyses in a consistent manner the parameter influence and allows a purposeful simplification and/or refinement of the model design.

14.2.3 Mechanical Engineering

Mehnen et al. (2007) apply SPOT in mechanical engineering for the design of *mold temperature control strategies* (MTCS), which is a challenging multi-objective optimization task. In this paper an EA is applied to a multi-objective problem using aggregation. The DACE technique is used to find good *multi-objective evolutionary algorithm* (MOEA) parameter settings to get improved solutions of the MTCS problem. An automatic and integrated software package, which is based on the DACE approach, is applied to find the statistically significant and most promising EA parameters using SPOT.

14.2.4 Biogas

Ziegenhirt et al. (2010) use SPOT to optimize the simulation of biogas plants. There is a high demand from industry to run these plants efficiently, which leads to a highly complex simulation and optimization problem. A comparison of several algorithms from computational intelligence to solve this problem is presented in their study. The sequential parameter optimization was used to determine improved parameter settings for these algorithms in an automated manner. They demonstrate that genetic algorithm and particle swarm optimization based approaches were outperformed by differential evolution and covariance matrix adaptation evolution strategy. Compared to previously presented results, their approach required only one tenth of the number of function evaluations.

14.2.5 Shipbuilding

Rudolph et al. (2009) discuss the optimization of a relatively new ship propulsion system (a linear jet) which possesses 15 design variables. It consists of a tube with a rotor and a stator, and several lengths, angles, and thicknesses that can be variated. The objective function is a very basic fluid dynamic simulation of a linear jet that takes about 3 minutes to compute, and the task is to reduce cavitation at a predefined efficiency. A full simulation would take about 8 hours, which is by far too much to serve as test problem. A surrogate model is used to detect good design points that can afterwards be validated by the full simulation. The authors conclude: "The validation experiment is successful, as the SPO-tuned parameter values lead to a significant performance increase. The tuned MAES resembles a model-enhanced (4,11)-ES with $\kappa = 20$ and so may profit from a more globally oriented behavior than the standard parameter setting with a population size of one."

14.2.6 Fuzzy Operator Trees

Yi (2008) proposes a method for modeling utility (rating) functions based on a novel concept called *fuzzy operator tree* (FOT). As the notion suggests, this method makes use of techniques from fuzzy set theory and implements a fuzzy rating function, that is, a utility function that maps to the unit interval, where 0 corresponds to the lowest and 1 to the highest evaluation. Even though the original motivation comes from quality control, FOTs are completely general and widely applicable. Yi (2008) reports that "several works have shown that SPO outperforms other alternatives, especially for evolution strategies, we shall apply SPO to determine the parameters of ES in this section. Sequential sampling approaches with adaptation have been proposed for DACE, here let us review the basic idea of Thomas Bartz-Beielstein, which is also used in this section to determine the parameters of ES."

14.3 Objectives

During the first stage of experimentation, SPOT treats the algorithm A as a black box. A set of input variables, say $\mathbf{x}$, is passed to A. Each run of the algorithm produces some output, $\mathbf{y}$. SPOT tries to determine a functional relationship F between $\mathbf{x}$ and $\mathbf{y}$ for a given problem formulated by an objective function $f : \mathbf{u} \rightarrow \mathbf{v}$. Since experiments are run on computers, pseudorandom numbers are taken into consideration if:

(i) The underlying objective function f is stochastically disturbed, e.g., measurement errors or noise occur, and/or

(ii) The algorithm A uses some stochastic elements, e.g., mutation in evolution strategies.

This situation can be described as follows:

$$\text{Objective function:} \quad \mathbf{u} \xrightarrow{f} \mathbf{v}, \tag{14.1}$$

$$\text{Algorithm:} \quad \mathbf{x} \xrightarrow{F} \mathbf{y}. \tag{14.2}$$

We will classify elements from (14.1) and (14.2) in the following manner, see also Sect. 2.2.1:

1. Variables that are necessary for the algorithm belong to the algorithm design, whereas
2. variables that are needed to specify the optimization problem f belong to the problem design.

SPOT employs a sequentially improved model to estimate the relationship between algorithm input variables and its output. This serves two primary goals. One is to enable determining good parameter settings, thus SPOT may be used as a tuner. Secondly, variable interactions can be revealed that help to understand how the tested algorithm works when confronted with a specific problem or how changes in the problem influence the algorithm's performance. Concerning the model, SPOT allows for insertion of virtually every available model. However, regression and Kriging models or a combination thereof are most frequently used.

14.4 Elements of the SPOT Framework

In this section, we describe SPOT as one possible implementation of steps 3a)-d) from the general SPO scheme (see Sect. 2.5.5.4) and discuss the different tools of the framework that may be employed separately or in automated mode. We conclude by giving a definition and a short introduction to starting and using SPOT.

14.4.1 The General SPOT Scheme

Definition 14.1 (Sequential Parameter Optimization Toolbox). The sequential parameter optimization toolbox implements the following features:

SPOT-1: Use the available budget (e.g., simulator runs, number of function evaluations) sequentially, i.e., use information from the exploration of the search space to guide the search by building one or several meta models. Choose new design points based on predictions from the meta model(s). Refine the meta model(s)) stepwise to improve knowledge about the search space.

SPOT-2: Try to cope with noise by improving confidence. Guarantee comparable confidence for search points.

SPOT-3: Collect information to learn from this tuning process, e.g., apply explorative data analysis.
SPOT-4: Provide mechanisms both for interactive and automated tuning.

□

Algorithm 14.1 presents a formal description of the SPOT scheme. This scheme consists of two phases, namely the first construction of the model and its sequential improvement. Phase 1 determines a population of initial designs in algorithm parameter space and runs the algorithm k times for each design. Phase 2 consists of a loop with the following components: By means of the obtained data, the model is built or updated, respectively. Then, a possibly large set of design points is generated and their predicted utility computed by sampling the model. A small set of the seemingly best design points is selected and the algorithm is run $k + 1$ times for each of these. The algorithm is also run once for the current best design point and k is increased by one. Note, other update rules for the number of repeats, k, are possible. The new design points are added to the population and the loop starts over if the termination criterion is not reached (usually a preset budget is granted to the process). In consequence, this means that the number of repeats is always increased by one if the current best design point stays at the top of the list or a newly generated one gets there. Due to nondeterministic responses of the algorithm, it may however happen that neither of these is found at the top of the list after finishing the loop. In this case, k may effectively shrink as performance comparisons have to be fair and thus shall be based on the same number of repeats.

Sequential approaches are generally more efficient, i.e., require fewer function evaluations, than approaches that evaluate the information in one step only. Chapter 15 of this book discusses an extension of this sequential framework. Further extensions were proposed by other authors, e.g., Lasarczyk (2007) integrated an *optimal computational budget allocation* (OCBA) procedure, which is based on ideas by Chen et al. (2003).

14.4.2 SPOT Tasks

If used for tuning an algorithm, SPOT is not per se envisioned as a meta-algorithm. We are interested in the resulting algorithm designs, not in the solutions to the primordial problem. However, it is of course possible to model a problem directly, without employing an algorithm. SPOT provides tools to perform the following tasks:

1. *Initialize*. An initial design is generated and written to a design file. This is usually the first step during experimentation. The employed parameter region and the constant algorithm parameters have to be provided by the user. A clean start can be performed. This is necessary if the user wants to run a new experiment or delete results from previous steps.

Algorithm 14.1: Sequential parameter optimization toolbox (SPOT)

```
//phase 1, building the model:
let $A$ be the tuned algorithm;
generate an initial population $X = \{\bar{x}^1, \dots, \bar{x}^m\}$ of $m$ parameter vectors;
let $k = k_0$ be the initial number of tests for determining estimated utilities;
foreach $\bar{x} \in X$ do
    run $A$ with $\bar{x}$ $k$ times to determine the estimated utility $y$ of $\bar{x}$;
end
//phase 2, using and improving the model:
while termination criterion not true do
    let $\bar{a}$ denote the parameter vector from $X$ with best estimated utility;
    let $k$ the number of repeats already computed for $\bar{a}$;
    build prediction model $f$ based on $X$ and $\{y^1, \dots, y^{|X|}\}$;
    generate a set $X'$ of $l$ new parameter vectors by random sampling;
    foreach $\bar{x} \in X'$ do
        calculate $f(\bar{x})$ to determine the predicted utility $f(\bar{x})$ of $\bar{x}$;
    end
    select set $X''$ of $d$ parameter vectors from $X'$ with best predicted utility ($d \ll l$);
    run $A$ with $\bar{a}$ once and recalculate its estimated utility using all $k + 1$ test results;
      //(improve confidence)
    let $k = k + 1$;
    run $A$ $k$ times with each $\bar{x} \in X''$ to determine the estimated utility $\bar{x}$;
    extend the population by $X = X \cup X''$;
end
```

2. *Run*. This is usually the second step. The optimization algorithm is started with configurations from the design file. The algorithm writes results to the result file. Information from the design and algorithm problem design files are used in this step.
3. *Sequential step*. A new design, based on information from the result file, is generated. This new design is written to the design file. A prediction model is used in this step. Several generic prediction models are available in SPOT already. To perform an efficient analysis, especially in situations when only few algorithms runs are possible, user-specified prediction models can easily be integrated into SPOT.
4. *Report*. An analysis, based on information from the result file, is generated. Since the report uses information from the result file, new report facilities can be added very easily. SPOT contains some scripts to perform a basic regression analysis and plots such as histograms, scatter plots, plots of the residuals, etc.
5. *Automatic* mode. In the automatic mode, the steps *run* and *sequential* are performed after the initialization a certain number of times.
6. *Meta* mode. In the meta mode, several problem instances can be used for tuning.

14.4.3 Running SPOT

Previous versions of the SPOT relied on functions provided by the MATLAB Kriging toolbox DACE developed by Lophaven et al. (2002). Starting with version 0.5, an R (Ihaka and Gentleman 1996) version is available, too. The following description refers to this R implementation (Bartz-Beielstein 2010b). Of course, an R-system must be available on your computer. The SPOT package can be obtained via CRAN, the *Comprehensive R Archive Network*, see `http://cran.r-project.org`. Therefore, the simplest way to install the package is to enter

```
install.packages("SPOT")
```

into your R session. SPOT can be loaded via

```
library("SPOT")
```

The formal command to start SPOT tasks reads:

```
spot( <configurationfile>, <task>)
```

where `task` can be one of the tasks described in Sect. 14.4.2, i.e., init, seq, run, rep, auto, or meta, and `configurationfile` is the name of the SPOT configuration

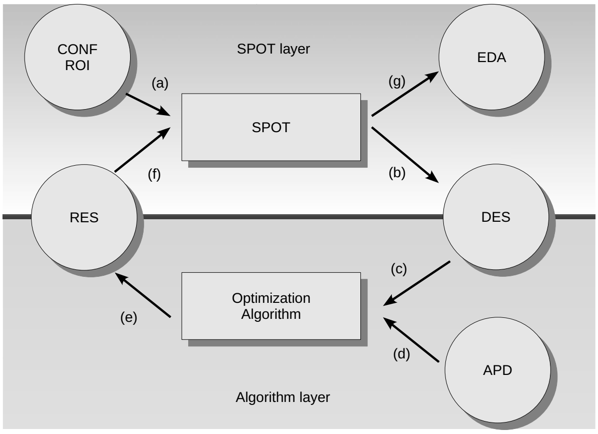


Fig. 14.1: SPOT interfaces. The SPOT loop can be described as follows: *Configuration* (CONF) and *region-of-interest* (ROI) files are read by SPOT (a). SPOT generates a *design* (DES) file (b). The algorithm reads the design file and (c) extra information, e.g., about the problem dimension from the *algorithm-problem design* (APD) file (d). Output from the optimization algorithm are written to the *result* (RES) file (e). The result file is used by SPOT to build the prediction model (f). Data can be used by *exploratory data analysis* (EDA) tools to generate reports, statistics, visualizations, etc. (g)

file. SPOT uses simple text files as interfaces to the algorithm to the statistical tools. A JAVA-based GUI, which simplifies parameter input, is available, too.

1. Files provided by the user:
   (a) *Region of interest* (ROI) files specify the region over which the algorithm parameters are tuned. Categorical variables such as the recombination operator in ES, are encoded as numerical values, e.g., "1" represents "no recombination" and "2" stands for "discrete recombination."
   (b) *Algorithm design* (APD) files are used to specify parameters used by the algorithm, e.g., problem dimension, objective function, starting point or initial seed.
   (c) *Configuration* files (CONF) specify SPOT specific parameters, such as the prediction model or the initial design size.
2. Files generated by SPOT:
   (a) *Design* files (DES) specify algorithm designs. They are generated automatically by SPOT and will be read by the optimization algorithms.
   (b) After the algorithm has been started with a parametrization from the algorithm design, the algorithm writes its results to the *result file* (RES). Result files provide the basis for many statistical evaluations/visualizations. They are read by SPOT to generate prediction models. Additional prediction models can easily be integrated into SPOT.

Figure 14.1 illustrates SPOT interfaces and the data flow. Note, that the problem design can be modified too. This can be done to analyze the robustness (effectivity) of algorithms.

SPOT can be run in an *automated* and an *interactive mode*. Similarities and differences of the automated and the interactive process are shown in Fig. 14.2.

14.5 Statistical Considerations

14.5.1 Sequential Models

To introduce SPOT's functionality, we describe a case study that analyzes the evolution strategy introduced in Chap. 2. Starting from scratch, we do not know anything about the functional relationship F from (14.1). This situation can be characterized as a chicken-and-egg problem: The experimenter has to decide which comes first: a design for the data $\mathbf{x}$ or the specification of a model, i.e., by defining a functional relationship F? For example, assuming a linear relationship F, we should use factorial designs to specify $\mathbf{x}$. However, in order to validate linearity, we need some data $\mathbf{x}$.

We prefer the following approach: Select a simple model (functional relationship) F_0 and a related design $\mathbf{x}_0$, generate some data $\mathbf{y}_0$, fit the (simple) model F_0,

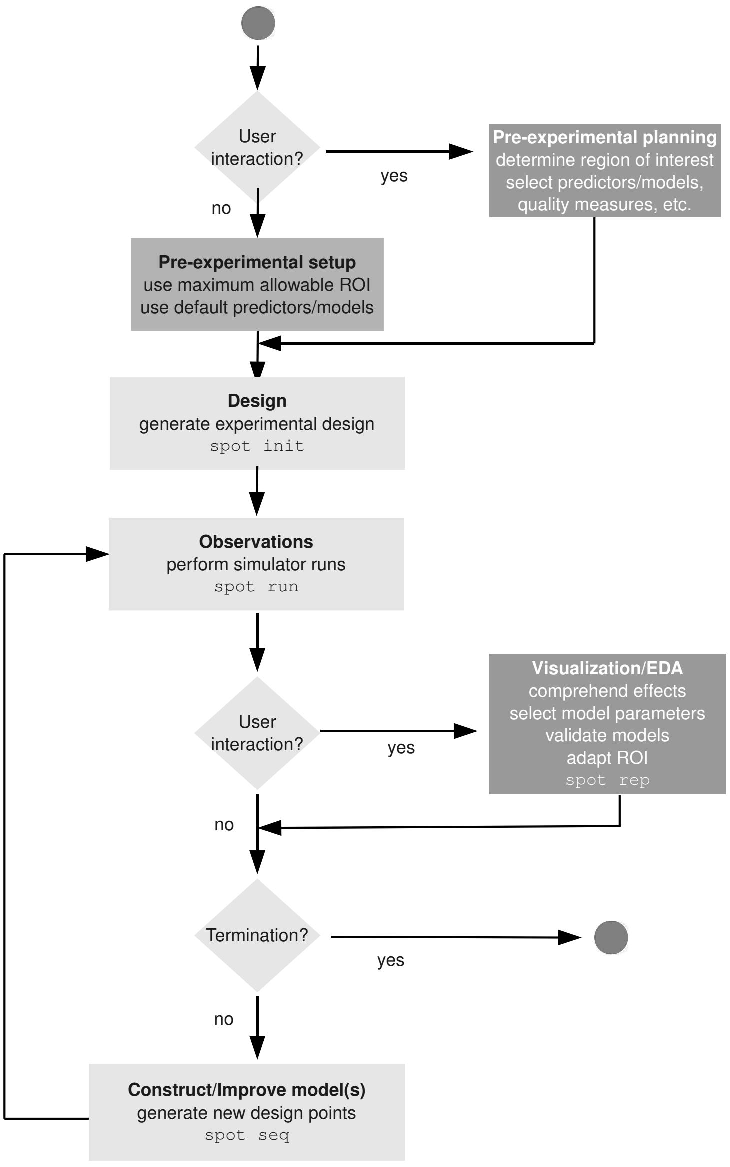


Fig. 14.2: The sequential parameter optimization process. *White font color* indicates elements that are used in the interactive process only. A *typewriter font* indicates the corresponding SPOT commands. To start the automated mode, simply use the task `auto`. Note that the interaction points are optional, so SPOT can be run without any user interaction

predict a few new design points $\mathbf{x}_1$ based on F_0, generate further data $\mathbf{y}_1$, refine the model (F_1), and continue. The chicken-and-egg problem defines a sequential approach in a natural manner, which improves F and y at the same time. This approach motivated the term *sequential parameter optimization*. In many situations, it can be shown that sequential approaches are much more efficient than one-at-a-time approaches (Armitage 1975).

We will discuss the basic output from the R analysis during the sequential approach. The analysis is not complicated and requires only basic knowledge of statistics. The reader will get interesting insights into the working mechanism of his algorithms by executing a few lines of R code. SPOT comes with some elementary R scripts which can be easily extended.

First, we will focus on fitting a linear regression model. The model

$$y = \beta_0 + \sum_{j=1}^{k} \beta_j x_j + \epsilon \tag{14.3}$$

is called a multiple linear regression model with k *regression variables* and *response* y. The regression variables represent algorithm parameters, e.g., initial step size s_0, multiplier for step sizes a, and size of the memory g. A variable that can be used to predict the value of another variable is also called an "input variable," "prediction variable," or "regressor." Response variables are also called "output variables." The corresponding predictive equation reads

$$\hat{\mathbf{y}} = \mathbf{b}\mathbf{X}. \tag{14.4}$$

The small roman letter $\mathbf{b}$ denotes a vector of the estimates of the parameters $\beta_0, \dots, \beta_k$.

R uses *Wilkinson–Rogers notation* to describe models, e.g.,

```
y ~ x1,
```

where y denotes the response, "$\sim$" can be read as "is modeled by," and $x1$ is the explanatory variable. Two or more explanatory variables can be incorporated into the model as shown in

```
y ~ x1 + x2.
```

There are more advanced ways of generating models, e.g., `A:B` for interactions, where `A:B` is a shorthand notation for `A+B+A*B`, etc.

Equation (14.3) describes a hyperplane in the k-dimensional space of the regressor variables x_j ($j = 1, \dots, k$). Assuming that all remaining independent variables x_i were held constant, the parameter β_j describes the expected change in y (response) per unit change in x_j ($i \neq j$).[1] We will discuss some considerations which are useful in the SPOT framework. The approach presented in this chapter relies on standard regression techniques only. Other models can be integrated into SPOT as well, e.g., tree-based regression or Kriging.

[1] A comprehensive introduction to regression is given in the appendix of this book.

Until now, no statistical assumptions that involve probability distributions have been made at all. In many situations, the following assumptions, which can be used to examine the regression coefficients, are made:

(a) the residual ϵ_i is a random variable with mean zero and unknown variance σ^2, i.e., $E(\epsilon_i) = 0$ and $V(\epsilon_i) = \sigma^2$,
(b) the ϵ's are uncorrelated, i.e., $\mathrm{Cov}(\epsilon_i, \epsilon_j) = 0$.
(c) $\epsilon_i \sim N(0, \sigma^2)$, i.e., ϵ_i is a normally distributed random variable, with mean zero and variance σ^2.

Assuming that the errors ϵ_i are all from the same normal distribution, the following test can be performed: The null hypothesis, i.e., β_1 equals β_{10}, especially $\beta_{10} = 0$, can be tested against the alternative that β_1 is different from β_{10} by calculating $t = \frac{b_1 - \beta_{10}}{\mathrm{se}(b_1)}$ (where $\mathrm{se}(b_1)$ denotes the standard error of the estimate of the parameter β_1) and comparing $|t|$ with $t(n-2, 1-\alpha/2)$, where $t(n, 1-\alpha)$ is the $100(1-\alpha)$ percentage point of a t-distribution with n degrees of freedom.

14.5.2 Residuals and Variance

The considerations in this section are not restricted to linear models only; they apply to any situation when a model is fitted. *Residuals* are, per definition, differences between values from actual observations and predictions by the regression equation. They describe the amount that the regression equation has not been able to explain. The residuals can be interpreted as the observed errors if the model is correct. We discuss graphical methods to examine residuals, especially plots of residuals versus fitted values, which could indicate that there should be a curve rather than a line, and normal probability plots of the residuals.

The mean square about the regression, s^2, will provide an estimate of the variance about the regression. If the regression equation (14.3) were estimated from an infinitely large number of experiments, the variance about the regression would provide a measure of the error with which any observed value of Y could be predicted from a given value of X using (14.3).

14.5.3 Standardized Variables and Transformations

The magnitude of the regression coefficients can be used to measure the effect of a variable. Problems can occur, because the regression coefficients depend on the underlying scale of measurements. For example, the coefficient for `time` measures the expected difference in response for each hour of difference in `time`. If `time` were measured in seconds instead of hours, the regression coefficient would be multiplied by 3,600, although the change in units does not change a variable's importance. To solve the problem of units of measurement, standardized regression coefficients can be used. We consider regression models with parameters β and standardized variables, i.e., each x_i ranges between -1 and $+1$. Although standardization appears appealing at the first sight, its usage is controversial; see, e.g., the discussion in Kleijnen (1987).

We have applied transformations on the function values, especially log-transformations, to improve the model fit. In general, we recommend using the raw data, e.g., to improve the interpretability of the results. In some situations, lack of fit was determined, to determine the most adequate model (linear versus quadratic etc.).

14.5.4 Design Considerations and the Region of Interest

Several designs can be used to generate initial data. Principally, two design types can be distinguished:

- Factorial designs, which are commonly used in classical regression to fit linear models, place design points at the border of the region of interest (Kleijnen 1987).
- Modern approaches such as Kriging use space-filling designs, e.g., Latin hypercube designs, which place design points in the interior of the region of interest (Santner et al. 2003).

The selection of an optimal design depends on the model which will be used for prediction. The pre-experimental planning phase includes test runs in a situation where no information about the model is available and no optimality conditions from DOE, e.g., as described in Pukelsheim (1993), are applicable. The main goal of this phase is to detect intervals for experimentation, i.e., the *region of interest* (ROI). As a rule of thumb, we can state that these intervals should be chosen courageously, because SPOT should be able to guide the search into promising regions of the search space. For some settings, the experimenter has to set up rules for the treatment of infeasible factor settings, e.g., by defining a penalty function or by introducing rounding mechanisms to ensure integer values.

Draper and Smith (1998, p. 86) discuss practical design-of-experiment implications of regression. They describe the influence of experimental arrangements for obtaining data for fitting a simple linear model. The question reads: At what values (sites) should how many experimental runs be performed? The relationship between lack of fit, pure error, $\mathrm{sd}(b_1)/\sigma$, and the number of sites is presented. They consider a situation in which n experiments can be performed and σ^2 is unknown. Performing n experiments at n different sites is a poor choice, because σ^2 cannot be estimated. On the other hand, performing $\lfloor n/2 \rfloor$ repeats at only two design sites results in no degrees of freedom left for estimating the lack of fit. On the basis of the $\mathrm{sd}(b_1)/\sigma$ values, configurations with many repeats are preferable. The best choice lies in arrangements that allow testing for lack of fit and minimization of the $\mathrm{sd}(b_1)/\sigma$ values.

Example 14.1. Suppose 14 runs are possible to fit a straight line. The range is the interval $[-1, +1]$ of the coded predictor. Variance σ^2 is unknown. Choosing 14 de-

Table 14.1: Regions of interest used in the pre-experimental planning phase. S0 and A denote real variables, whereas G is an integer value

Factor	Low	High	Type
S0	0.1	5	FLOAT
A	0	2	FLOAT
G	1	100	INT

sign points with only one repeat at each point is a poor choice, because σ^2 cannot be estimated. However, also the use of seven repeats at two design points is not recommended, because lack of fit (to test the adequacy of a linear model) cannot be checked (Draper and Smith 1998). □

14.6 A Case Study: Simple Regression Applied to Evolution Strategies

Classical *response surface methods* (RSM) use three steps: screening, modeling, and optimization (Kleijnen 1987, Montgomery 2001). Design complexity is increased during the RSM process, so complex models can be fitted in the last phase of the RSM process. SPOT is closely related to RSM. Because SPOT includes rules to analyze the scientific relevance (severity) of results from the statistical analysis and specifies rules for learning from error, it can be seen as an extension of the classical RSM framework.

14.6.1 Pre-experimental Planning

This section describes how basic ideas from regression analysis can be applied to analyze evolution strategies. SPOT allows the specification of:

a) upper and lower limits of the region of interest (experimental region),
b) the number of repeats for the initial design, and
c) the type of design to be generated, e.g., LHD.

We have chosen a Latin hypercube design for the first experiments, see Table 14.1. This design consists of three factors with different types. Although factorial designs are recommended for DOE (see also the discussion in Sect. 14.5.4), a space-filling design was chosen for the pre-experimental planning phase. This design was chosen because we expect interesting model features in the center of the experimental region, and we are interested in using other models, e.g., tree-based regression models, or Kriging, in parallel. These models require space-filling designs. So, choosing

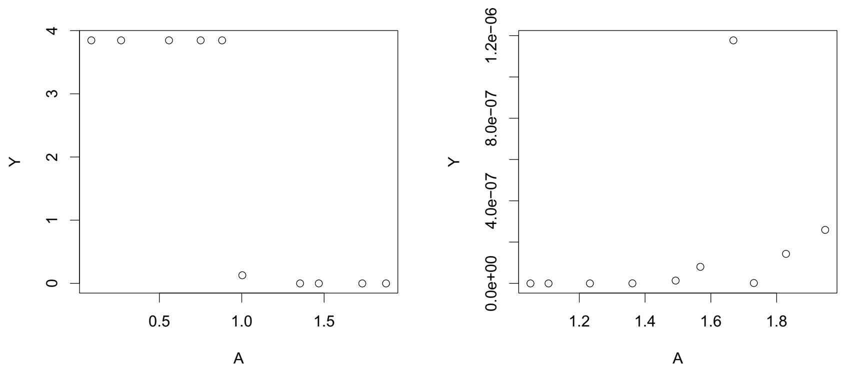


Fig. 14.3: Scatter plots from the pre-experimental planning phase to determine the region of interest. *Left:* A range from 0 to 2 for the multiplier a was chosen. Values smaller than 1 invert the 1/5th rule (2.4). As can be seen from the figures, values smaller than 1 worsen the performance of the (1+1)-ES. *Right:* If a is chosen from the interval between 1 and 2, the (1+1)-ES performs quite reasonably. These results support the recommendations given in (2.4)

an LHD can be seen as a good compromise, especially if we do not know whether linear regression models are adequate.

A *scatter* plot was used to analyze results from the first design. It is based on ten design points, where each design point represents one algorithm configuration. Each run was performed once. Figure 14.3 clearly illustrates that the multiplier for the step length should be larger than 1.0. This supports the assumption that step sizes should be increased if the success rate is larger than $1/5$. Values smaller than 1.0 are excluded from the following investigations. By repeating the same configuration as in the first study, but with values for A chosen from the interval $[1, 2]$, significant improvements can be observed. The algorithm performs quite well with parameters chosen from these ROI. Therefore, we decided to start the analysis with a multiplier of the step width chosen from the interval $[1, 2]$.

14.6.2 Performing the First Regression Analysis

To regress fitness, y, on the these quantitative predictors, we obtain the output in R as shown in Fig. 14.4.

The R output gives a numerical summary of the residuals and a table of the regression parameters. For example, the coefficient b_1 is the change in the response y when x_1 is changed by one unit. Besides estimated regression coefficients and their standard errors, t-statistics and p-values for the individual tests of the hypotheses are included. Here, we see that the intercept of the fitted line is $b_0 = -28.0456$

```
Call:
lm(formula = log(Y) ~ log(S0) + log(A) + log(G), data = df0002)
Residuals:
    Min      1Q  Median      3Q     Max
-3.6093 -1.7175  0.7059  1.4426  3.0768
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -28.0456     5.6820  -4.936  0.00261 **
log(S0)       0.4115     0.9836   0.418  0.69022
log(A)       27.2698     4.8056   5.675  0.00129 **
log(G)       -0.9124     1.2219  -0.747  0.48346
---
Residual standard error: 2.791 on 6 degrees of freedom
Multiple R-squared: 0.8671,Adjusted R-squared: 0.8006
F-statistic: 13.05 on 3 and 6 DF,  p-value: 0.004872
```

Fig. 14.4: Output from the first regression model in R.

with se$(b_0) = 5.6820$, and the estimated regression coefficient b_1 is 0.4115 with se$(b_1) = 0.9836$. The multiplier A has the largest effect, because its value reads 27.2698. The memory G has only minor impact on the performance of the ES. The table reports also t-statistics and p-values for individual tests of the hypotheses that the true intercept is zero and that the true slope is zero. The `Residual standard error` is an expression of the variation of the observations around the fitted line. It can be used to estimate σ. `Multiple R-Squared` and `Adjusted R-Squared` are reported as well. The values related to the `F-statistic` describe an F-test for the hypothesis that the regression coefficients are zero.

In addition, an *analysis of variance* (ANOVA) table can be generated. Based on an ANOVA table we can compare two models:

(a) Model M_1, which includes $S0$, A, and G
(b) Model M_2, which includes the parameter with the highest significance, namely A

As expected, this indicates that there is no significant improvement of the model once $S0$ and G are included. Removing the parameters $S0$ and G from the model is recommended. R has a built-in function which adds parameters one at a time to the current model. The `add1` function adds parameters one after another from a list and shows the resulting statistics. The default output table reports the *Akaike information criterion* (AIC), defined as minus twice the log likelihood plus $2p$, where p is the rank of the model (the number of effective parameters). For performing model searches by AIC, R provides the `stepAIC` function. Since selection and adjustment of information criteria is a difficult task (and beyond the scope of this introduction), we simply show the output from `stepAIC` applied to our example in Fig. 14.5.

The final model, suggested by `stepAIC`, includes the parameter A only.

Residual plots can be used to check model assumptions. Common checks include:

```
Call:
lm(formula = Y ~ A, data = df0002normlogy)
Residuals:
    Min      1Q  Median      3Q     Max
-3.7071 -2.6383  0.2352  2.4411  3.8783
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -20.6070     0.9672 -21.305 2.48e-08 ***
A             8.1253     1.4940   5.439 0.000617 ***
---
Residual standard error: 3.059 on 8 degrees of freedom
Multiple R-squared: 0.7871,Adjusted R-squared: 0.7605
F-statistic: 29.58 on 1 and 8 DF,  p-value: 0.000617
```

Fig. 14.5: Output from the model based on R's `stepAIC` function

a) A plot of residuals versus fitted values
b) Normal probability plots of residuals

In addition, visual inspections, e.g., on the basis of *added-variable plots* (Fig. 14.6) can be performed. Added-variable plots focus on one variable at a time and take into account the influence of the other predictors (Draper and Smith 1998, Fox 2002). To determine the influence of predictor x_j on the response y, added-variable plots are generated as follows. Let x_{-i} denote the set of regressors

$$\{x_1, x_2, \dots, x_{i-1}, x_{i+1}, \dots, x_n\}.$$

1. Regress y on x_{-j} , obtaining residuals $e(j)$.
2. Regress x_j on predictors x_{-j} , obtaining residuals $x(j)$.
3. Plot $e(j)$ versus $x(j)$.

All ES runs were performed using the same number of function evaluations. The results clearly indicate that A should be reduced. Smaller values improve the algorithms performance. The linear model was able to detect this trend. This was the first step of the analysis. The experimental setup shown in Table 14.1 includes a systematic variation of three parameters.

Summarizing Results from the Regression Model Analysis

The regression models used in this study illustrate that A should be decreased. Stepwise regression results in the simplified model

$$\hat{y} = -20.61 + 8.12a,$$

which indicates that the multiplier for the step sizes should be decreased. A similar result was obtained from a tree-based regression. The result occurs independently from the chosen model.

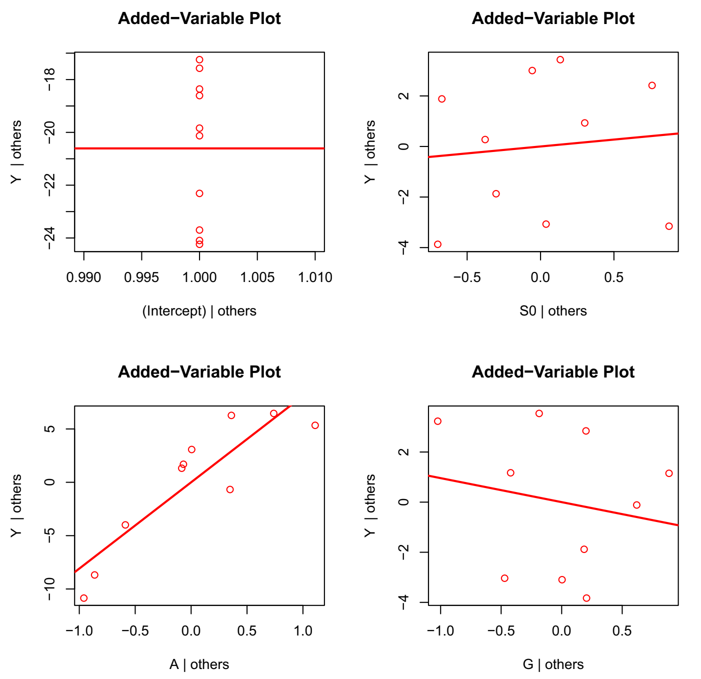


Fig. 14.6: Added-variable plot based on ten runs from the initial design of ES on the ten-dimensional sphere function. Smaller y values represent better solutions

How should one proceed if interactions that play an important role in the regression model might be of interest? We recommend *not* to apply the method of steepest descent based on the main factor model if there are significant interactions in the model. Instead, the optimum predicted by the reduced regression model including the main effects and the interactions can be used (Mehnen et al. 2004).

14.6.3 Steepest Descent

Results from the regression analysis provide information about the steepest descent in a natural manner. We proceed with the steepest descent based on the model `Y ~ S0 + A + G`. The procedure of *steepest descent* is performed as follows: Starting from the design center, we move sequentially in the direction of the maximum decrease in the response. This direction is parallel to the normal of the fitted response surface. Usually in RSM, the path of the steepest descent is taken as the line through the center of the ROI and normal to the fitted surface, i.e., the steps are proportional to the β_i's (regression coefficients). In general, the following procedure

Table 14.2: Steepest descent. $S0$ and A denote real variables, whereas G is an integer value, so its values are rounded to the next integer

	S0	A	G
1	2.54	1.50	52.50
2	2.52	1.45	53.05
3	2.50	1.41	53.60
4	2.49	1.37	54.15
5	2.47	1.32	54.70
6	2.45	1.28	55.25
7	2.44	1.23	55.80
8	2.42	1.19	56.35
9	2.40	1.14	56.90
10	2.39	1.10	57.45
11	2.37	1.05	58.01

for determining the coordinates of the points on the path of the steepest descent can be applied (Montgomery 2001):

1. Select the variable we know most about or the variable that has the largest absolute regression coefficient $|b_i|$.
2. Determine the step size in the other variables as
$$\Delta x_i = \frac{b_i}{b_j}\Delta x_j \qquad i = 1, 2, \ldots, k; \qquad i \neq j.$$
3. Convert the Δx_i stepwidths from the coded to the δx_i in the natural variables.

As the largest step size we recommend the value that leads to the border of the ROI. Data from the steepest descent experiments are shown in Table 14.2. Note that the steepest-descent experiments do not have to be performed in sequence. For example, run 10 can be performed before run 2, or a simple interval search can be performed. We recommend generating a graph of these results to determine the new region of interest following the direction of the steepest descent, cf. Fig. 14.7. The region of interest is moved down the response surface.

These settings are used for additional runs of ES. Figure 14.7 plots the yield at each step along the path of the steepest descent.

Based on visual inspection of the yields in Fig. 14.7, the new central point was determined to be the tenth point of the steepest descent. This new central point leaves some space for variation of the A values, say in the interval $[1.01, 1.2]$. Based on the best value obtained with the steepest descent, we build a new model with center point
$$\mathbf{x}_c = [S0, A, G] = [2.5, 1.125, 40].$$

The specification of the new region of interest requires user knowledge. The new center point was determined by interpreting graphical results which were based on the steepest descent. Next, we have to determine a new region around $\mathbf{x}_c$. Sometimes, especially when a classical factorial design is used during the first step, it can

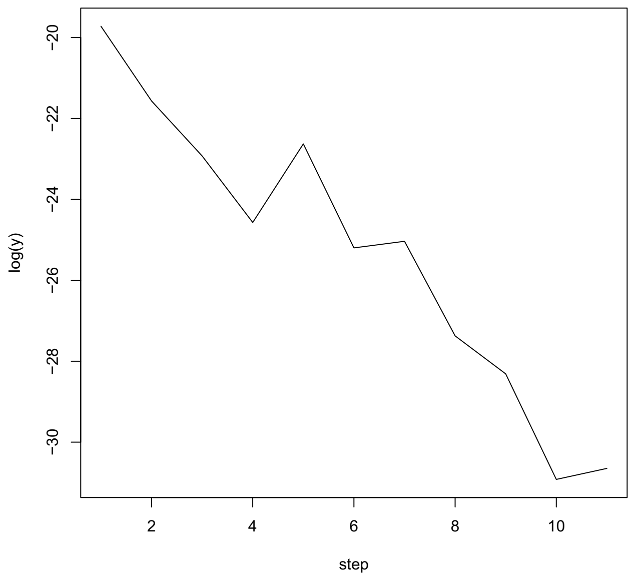


Fig. 14.7: Function values $f(x)$ (sphere) versus steps along the path of the steepest descent. Indices denote the ten steps from the steepest descent. Note, that these values are based on one repeat only, so variation in the data, e.g., the peak (index 5), is not surprising

be useful to increase the region of interest at this stage. However, we have chosen a space-filling design for the first step. Therefore, we have to decrease the region of interest. As a rule of thumb, which is to be reconsidered in any situation, we use at least $\pm\ 1/5$th of the values at the new central point. For example, if the value of the new initial step size $S0$ is 5, we define a new region of interest for this values as the interval $[4,6]$. Here, the new region of interest reads as follows:

$$S0 \in [2,3], A \in [1.05,2], G \in [20,80].$$

Latin hypercube sampling with 100 points and three repeats each is used for a graphical exploration of the ROI. Figure 14.8 displays a fit of the response surface which is based on the complete data set (320 runs of the target algorithm). A local regression model based on R's `loess()` function is fitted to the data.

Summarizing Results from the RSM

Results from this case study support results presented by Beyer (2001). The factor a should be chosen in the range from 1.1 to 1.5. Further experiments show that this

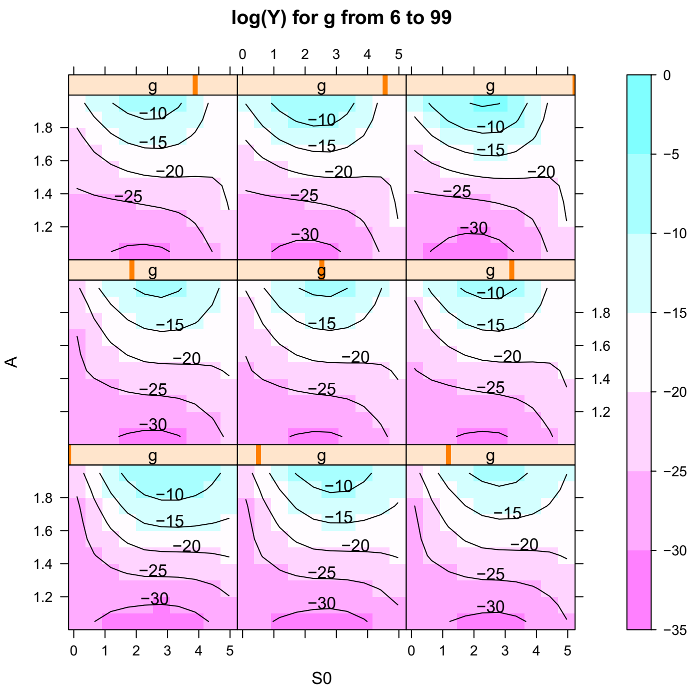


Fig. 14.8: Contour plot based on the complete data set (ES-Sphere with 320 function evaluations). Smaller values are better. Better configurations are placed in the lower-left corner of the panels. A is plotted versus $S0$, while values of the factor with the last but one effect, namely G, are varied with the slider on top of each panel

result is independent of the starting point and problem dimension. The corresponding contour plots indicate that runs of the (1+1)-ES with values for a chosen in the recommended interval result in good function values. The structure of the contour plots does not change if starting point or problem dimension are modified.

14.7 Additional Model Considerations

The analysis of optimization algorithms requires the investigation of categorical variables; e.g., in evolution strategies, several variants of the recombination operator can be used (Beyer and Schwefel 2002). SPOT allows the coding of nonnumerical variables as factors. *Dummy regressors* or *contrasts* can be used to represent levels of a factor. SPOT can handle the following variables:

1. real values,
2. integers, and

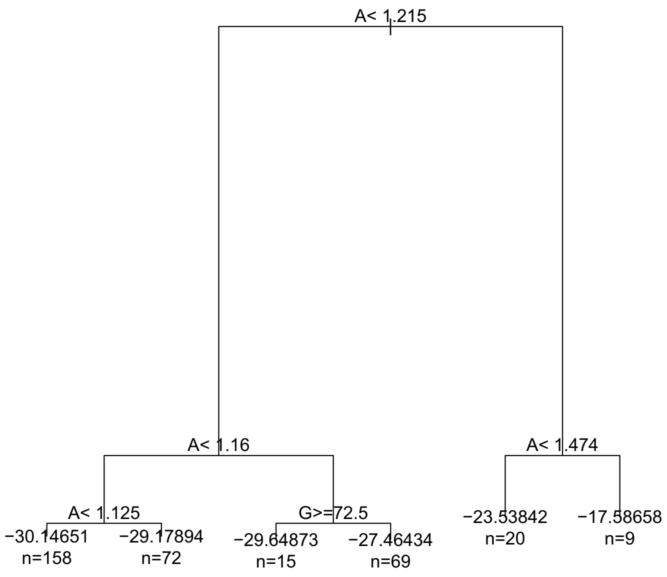


Fig. 14.9: Tree-based regression. The same data as in Fig. 14.8 were used. The factor A has the largest effect. $\log(y)$ values were used to grow the tree

3. categorical variables.

SPOT's ROI file is used to specify this typing. Maindonald and Braun (2003) illustrate the treatment of dummy variables for classical regression in R.

Classical regression is only one technique that can be used in the SPOT framework to predict interesting design points. Alternatively, *tree-based models* can be used to cope with categorical variables; see also Fig. 14.9. Tree-based methods can be used for regression as well as for classification (Breiman et al. 1984). Maindonald and Braun (2003) recommend to use tree-based methods in tandem with parametric approaches, because tree-based regression may suggest interaction terms that ought to appear in a parametric model. However, tree-based methods require more data than classical regression techniques. That is, if only a few runs of the algorithms are possible, it may be necessary to use parametric models. On the other hand, tree-based methods may be helpful to explore new data sets very quickly: the experimenter gets on overview of which variables have the greatest effects on the algorithm's performance.

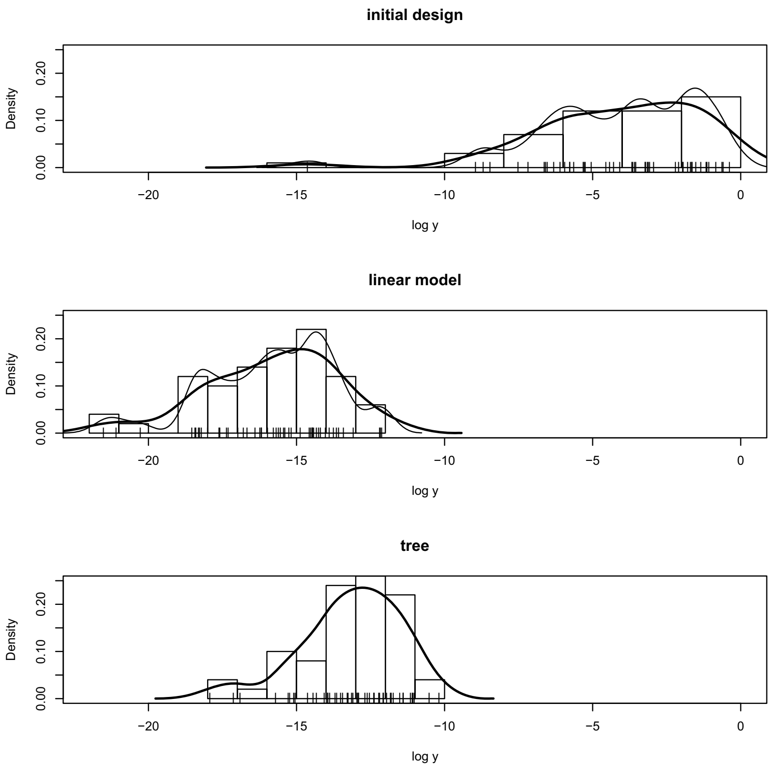


Fig. 14.10: Comparison of two SPOT prediction models with randomly generated configurations (from *top* to *bottom*). First, the results from randomly generated algorithm designs are shown. We have chosen the initial design (LHD), which is generated randomly. Next, results from 50 repeats of the best algorithm design determined with the linear and the tree-based regression model are displayed

We have mentioned only two prediction models: parametrized regression and regression trees. Further models, e.g., local regression, Kriging or mixed models as introduced in Sect. 10.1, are also available.

Example 14.2. To analyze the choice of the prediction model on the prediction quality and SPOT's ability to improve optimization algorithms (tuning), we performed the following study: Results from two different SPOT runs are compared. The first run uses a tree-based regression model, whereas in the second run the dummy-variable regression model was used. Both models used exactly the same setting, i.e., five SPOT iterations, initial design size of 50 points, and two repeats. An ES, which optimizes the ten-dimensional sphere function with 1,000 function evaluations was analyzed. Distributions of the results from these two SPOT runs are compared with the distribution of the randomly generated initial design in Fig. 14.10. Both models were able to find better results than the randomized design. These results are in line with observations from other studies: Tree-based models can be applied very easily to unknown explanatory variables. They can cope with categorical and numerical data. Parametrized models perform better than tree-based models; however, the costs for modeling are higher. □

14.8 The Automated Mode

In the case study in Sect. 14.6 we discussed the initial setup for the SPOT loop (initial design) and the analysis of one step. SPOT can proceed as follows: Based on the prediction model, e.g., linear regression or tree-based regression, interesting algorithm design points are generated. These design points are evaluated, i.e., the algorithm is run with the corresponding parameters. Then, an analysis as described in Sect. 14.6.2 can be performed. This analysis provides an improved predictor, which can be used to propose new design points, and so forth. As depicted in Fig. 14.2, this procedure can be performed in an automated manner. The result from the automated approach reads

$$s_0 = 4.99, a = 1.10, g = 71.$$

Obviously, the result from the automated approach supports the findings from the manual approach, i.e., similar values for the parametrization of the (1+1)-ES are determined.

14.9 Summary

In this chapter, basic elements of the SPOT framework were introduced and discussed. SPOT is an open-source implementation of the sequential parameter optimization presented in Chap.2. It requires the specification of the region of interest, the algorithm design, and SPOT-related configuration parameters.

SPOT provides tools to perform the following four elementary tasks: (i) generate an initial design, (ii) run the algorithm, (iii) generate a new design based on results from previous runs, and (iv) generate a report. Additionally, these tasks can be run in an automated mode.

We demonstrated how a simple analysis of the (1+1)-ES can be performed. This analysis requires the specification of the following elements:

1. Region of interest, i.e., the range of parameter settings of the algorithm.
2. Algorithm- and problem-related parameters.
3. Metaparameters used by SPOT.

A simple regression model, well known in DOE and RSM (Montgomery 2001), was used to analyze the influence (importance) of three parameters of the (1+1)-ES, namely, initial stepsize, multiplier for the stepsize, and the length of the memory vector. The experimental results support a result which was derived theoretically by Beyer (2001).

SPOT is being developed, applied, and improved at several research institutes around the world. By providing an open-source implementation and a graphical user interface, we hope that SPOT can be a useful tool for the research community.

Acknowledgements This work has been supported by the Bundesministerium für Forschung und Bildung (BMBF) under the grant FIWA (AIF FKZ 17N2309, "Ingenieurnachwuchs") and by the Cologne University of Applied Sciences under the grant COSA.

References

Armitage P (1975) Sequential medical trials, 2nd edn. Blackwell, Oxford, U.K.

Bartz-Beielstein T (2010a) Sequential parameter optimization. Tech. Rep. 04/2010, Institute of Computer Science, Faculty of Computer Science and Engineering Science, Cologne University of Applied Sciences, Germany, URL `http://www.gm.fh-koeln.de/imperia/md/content/personen/lehrende/bartz_beielstein_thomas/spotannotatedbib.pdf`

Bartz-Beielstein T (2010b) SPOT: An R package for automatic and interactive tuning of optimization algorithms by sequential parameter optimization. Tech. Rep. CIOP TR 05-10, Cologne University of Applied Sciences, URL `http://arxiv.org/abs/1006.4645`, related software can be downloaded from http://cran.r-project.org/web/packages/SPOT/index.html

Bartz-Beielstein T, Lasarczyk C, Preuß M (2005) Sequential parameter optimization. In: McKay B, et al. (eds) Proceedings 2005 Congress on Evolutionary Computation (CEC'05), Edinburgh, Scotland, IEEE Press, Piscataway NJ, vol 1, pp 773–780

Beyer HG (2001) The Theory of Evolution Strategies. Springer

Beyer HG, Schwefel HP (2002) Evolution strategies—A comprehensive introduction. Natural Computing 1:3–52

Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and Regression Trees. Wadsworth, Monterey CA

Chen J, Chen C, Kelton D (2003) Optimal computing budget allocation of indifference-zone-selection procedures, working paper, taken from `http://www.cba.uc.edu/faculty/keltonwd`. Cited 6 January 2005

Draper NR, Smith H (1998) Applied Regression Analysis, 3rd edn. Wiley, New York NY

Fober T, Mernberger M, Klebe G, Hüllermeier E (2009) Evolutionary construction of multiple graph alignments for the structural analysis of biomolecules. Bioinformatics 25(16):2110–2117

Fox J (2002) An R and S-Plus Companion to Applied Regression. Sage

Ihaka R, Gentleman R (1996) R: A language for data analysis and graphics. Journal of Computational and Graphical Statistics 5(3):299–314

Kleijnen JPC (1987) Statistical Tools for Simulation Practitioners. Marcel Dekker, New York NY

Konen W, Zimmer T, Bartz-Beielstein T (2009) Optimierte Modellierung von Füllständen in Regenüberlaufbecken mittels CI-basierter Parameterselektion Optimized Modelling of Fill Levels in Stormwater Tanks Using CI-based Parameter Selection Schemes. at-Automatisierungstechnik 57(3):155–166

Lasarczyk CWG (2007) Genetische Programmierung einer algorithmischen Chemie. PhD thesis, Technische Universität Dortmund

Lophaven S, Nielsen H, Søndergaard J (2002) DACE—A Matlab Kriging Toolbox. Tech. Rep. IMM-REP-2002-12, Informatics and Mathematical Modelling, Technical University of Denmark, Copenhagen, Denmark

Maindonald J, Braun J (2003) Data Analysis and Graphics using R—an Example-based Approach. Cambridge University Press, Cambridge UK

Mehnen J, Michelitsch T, Bartz-Beielstein T, Henkenjohann N (2004) Systematic analyses of multi-objective evolutionary algorithms applied to real-world problems using statistical design of experiments. In: Teti R (ed) Proceedings Fourth International Seminar Intelligent Computation in Manufacturing Engineering, CIRP ICME'04, Naples, Italy, vol 4, pp 171–178

Mehnen J, Michelitsch T, Lasarczyk C, Bartz-Beielstein T (2007) Multi-objective evolutionary design of mold temperature control using DACE for parameter optimization. International Journal of Applied Electromagnetics and Mechanics 25(1–4):661–667

Montgomery DC (2001) Design and Analysis of Experiments, 5th edn. Wiley, New York NY

Pukelsheim F (1993) Optimal Design of Experiments. Wiley, New York NY

Rudolph G, Preuss M, Quadflieg J (2009) Two-layered surrogate modeling for tuning optimization metaheuristics. Algorithm Engineering Report TR09-2-005, Faculty of Computer Science, Algorithm Engineering (Ls11), Technische Universität Dortmund, Germany

Santner TJ, Williams BJ, Notz WI (2003) The Design and Analysis of Computer Experiments. Springer

Tukey J (1991) The philosophy of multiple comparisons. Statistical Science 6:100–116

Volkert L (2006) Investigating ea based training of hmm using a sequential parameter optimization approach. In: Yen GG, Lucas SM, Fogel G, Kendall G, Salomon R, Zhang BT, Coello CAC, Runarsson TP (eds) Proceedings of the 2006 IEEE Congress on Evolutionary Computation, IEEE Press, Vancouver, BC, Canada, pp 2742–2749, URL `http://ieeexplore.ieee.org/servlet/opac?punumber=11108`

Yi Y (2008) Fuzzy operator trees for modeling utility functions. PhD thesis, Philipps-Universität Marburg

Ziegenhirt J, Bartz-Beielstein T, Flasch O, Konen W, Zaefferer M (2010) Optimization of biogas production with computational intelligence—a comparative study. Tech. Rep. 03/2010, Institute of Computer Science, Faculty of Computer Science and Engineering Science, Cologne University of Applied Sciences, Germany

Chapter 15
Sequential Model-Based Parameter Optimization: an Experimental Investigation of Automated and Interactive Approaches

Frank Hutter, Thomas Bartz-Beielstein, Holger H. Hoos, Kevin Leyton-Brown, and Kevin P. Murphy

Abstract This work experimentally investigates model-based approaches for optimizing the performance of parameterized randomized algorithms. Such approaches build a response surface model and use this model for finding good parameter settings of the given algorithm. We evaluated two methods from the literature that are based on Gaussian process models: *sequential parameter optimization* (SPO) (Bartz-Beielstein et al. 2005) and *sequential Kriging optimization* (SKO) (Huang et al. 2006). SPO performed better "out-of-the-box," whereas SKO was competitive when response values were log transformed. We then investigated key design decisions within the SPO paradigm, characterizing the performance consequences of each. Based on these findings, we propose a new version of SPO, dubbed SPO$^+$, which extends SPO with a novel intensification procedure and a log-transformed objective function. In a domain for which performance results for other (model-free) parameter optimization approaches are available, we demonstrate that SPO$^+$ achieves state-of-the-art performance. Finally, we compare this automated parameter tuning approach to an interactive, manual process that makes use of classical

Frank Hutter
Department of Computer Science, University of British Columbia, 201-2366 Main Mall, Vancouver BC, V6T 1Z4, Canada, e-mail: hutter@cs.ubc.ca

Thomas Bartz-Beielstein
Institute of Computer Science, Cologne University of Applied Sciences, 51643 Gummersbach, Germany, e-mail: thomas.bartz-beielstein@fh-koeln.de

Holger H. Hoos
Department of Computer Science, University of British Columbia, 201-2366 Main Mall, Vancouver BC, V6T 1Z4, Canada, e-mail: hoos@cs.ubc.ca

Kevin Leyton-Brown
Department of Computer Science, University of British Columbia, 201-2366 Main Mall, Vancouver BC, V6T 1Z4, Canada, e-mail: kevinlb@cs.ubc.ca

Kevin P. Murphy
Department of Computer Science, University of British Columbia, 201-2366 Main Mall, Vancouver BC, V6T 1Z4, Canada, e-mail: murphyk@cs.ubc.ca

regression techniques. This interactive approach is particularly useful when only a relatively small number of parameter configurations can be evaluated. Because it can relatively quickly draw attention to important parameters and parameter interactions, it can help experts gain insights into the parameter response of a given algorithm and identify reasonable parameter settings.

15.1 Introduction

Many high-performance algorithms—and, in particular, many heuristic solvers for computationally challenging problems—expose parameters to allow end users to adapt the algorithm to target applications. Optimizing parameter settings is thus an important task in the context of developing, evaluating and applying such algorithms. Recently, a substantial amount of research has been aimed at defining effective, automated procedures for parameter optimization (also called *algorithm configuration* or *parameter tuning*). More specifically, the goal is to find parameter settings of a given target algorithm that optimize a given performance metric on a given set (or distribution) of problem instances. The performance metric is usually based on the runtime required to solve a problem instance or—in the case of optimization problems—on the solution quality achieved within a given time budget.

Several variations of this problem have been investigated in the literature. These formulations vary in the number and type of target algorithm parameters allowed. Much existing work deals with relatively small numbers of numerical (often continuous) parameters; see, e.g., Coy et al. (2001), Audet and Orban (2006), Adenso-Diaz and Laguna (2006). Some relatively recent approaches permit both larger numbers of parameters and categorical domains; see, e.g., Birattari et al. (2002), Beielstein (2003), Bartz-Beielstein and Markon (2004), Bartz-Beielstein et al. (2004c), Hutter et al. (2007, 2009b). A different problem formulation also permits parameter adaptation on a per-instance basis; see, e.g., Hutter et al. (2006).

Approaches also differ in whether or not explicit models (so-called *response surfaces*) are used to describe the dependence of target algorithm performance on parameter settings. There has been a substantial amount of work on both model-free and model-based approaches. Some notable model-free approaches include F-Race by Birattari et al. (2002), Balaprakash et al. (2007), CALIBRA by Adenso-Diaz and Laguna (2006), and ParamILS by Hutter et al. (2007). State-of-the-art model-based approaches use Gaussian stochastic processes (also known as Kriging models) to fit a response surface model. These models aim to minimize the mean squared error between predicted and actual responses, using a nonparametric function to represent the mean response. One particularly popular and widely studied version of Gaussian stochastic process models in the statistics literature is known under the acronym DACE, for *design and analysis of computer experiments* (Sacks et al. 1989). Combining such a predictive model with sequential decisions about the most promising next design point (often based on a so-called *expected improvement criterion*) gives rise to a sequential optimization approach. An influential contribution in this field was the *efficient global optimization* (EGO) procedure by Jones et al. (1998), which addressed the optimization of deterministic black-box functions. In the context of parameter optimization, EGO could be used to optimize deterministic algorithms with continuous parameters on a single problem instance. Two independent lines of work extended EGO to noisy functions, which in the context of parameter optimization, allow the consideration of randomized algorithms: the *sequential Kriging optimization* (SKO) algorithm by Huang et al. (2006), and the *sequential parameter optimization* (SPO) procedure by Bartz-Beielstein et al. (2004b, 2005).

In the first part of this chapter, we maintain this focus on *Gaussian process* (GP) models; while in the second part, we consider the use of classical models in an interactive approach. Throughout, we limit ourselves to the simple case of only one problem instance. Such an instance may be chosen as representative of a set or distribution of similar instances. This restriction allows us to sidestep problems arising from performance variation across a set or distribution of problem instances and to focus on other core conceptual issues, while retaining significant practical relevance. (The management of such variation is an interesting and important topic of study; indeed, we have already begun to investigate it in our ongoing work. We note that this problem can be addressed by the algorithm of Williams et al. (2000), though only in the case of deterministic target algorithms.)

This chapter leverages our own previous work in a number of ways. In particular, a short version of Sects. 15.1–15.5 was published by Hutter et al. (2009a). Here, we formalize mathematical and algorithmic concepts from that paper much more thoroughly and provide more details throughout. The general sequential optimization approach using Kriging models has a long tradition in the statistics literature (Mockus et al. 1978, Jones et al. 1998) and was adapted to the optimization of algorithm parameters by Bartz-Beielstein et al. (2004a,b). Bartz-Beielstein et al. (2005) summarized results from SPO applications in various problem domains and Bartz-Beielstein (2006) described SPO's methodology in detail. Bartz-Beielstein and Preuss (2006) and Bartz-Beielstein et al. (2008b) studied the allocation of a fixed computational budget to SPO's initial design and to the evaluation of each parameter setting. The interactive approach used in Sect. 15.6 was first presented by Beielstein (2003), Bartz-Beielstein (2003), and Bartz-Beielstein and Markon (2004).

In the following, we use the term *sequential parameter optimization* (SPO) to refer to the methodological framework, i.e., a sequential approach to improve and understand an algorithm's performance by optimizing its parameters. The *sequential parameter optimization toolbox* (SPOT) is a software framework supporting both automated and interactive applications of this approach. Further discussion of SPOT is offered in Chap. 14, while the general SPO framework is presented in Chap. 2. Finally, we study three fully-automatic SPO procedures, which we refer to as SPO 0.3, SPO 0.4, and SPO$^+$. In the first part of the chapter, where we study these procedures in detail, we sometimes also use the term SPO to refer to the algorithmic framework of which SPO 0.3, SPO 0.4, and SPO$^+$ are instantiations.

The first part of our study thoroughly investigates the two fundamental components of any model-based optimization approach in this setting: the procedure for building the predictive model and the sequential procedure that uses this model to find performance-optimizing parameter settings of the target algorithm. We begin in Sect. 15.2 by describing our experimental setup, focusing especially on the two target algorithms we consider: CMA-ES (Hansen and Ostermeier 1996, Hansen and Kern 2004), a prominent gradient-free numerical optimization algorithm, and SAPS (Hutter et al. 2002), a high-performance local search algorithm for the propositional satisfiability problem. In Sect. 15.3, we compare the model-based optimization procedures SKO and SPO. Overall, we found that SPO produced more robust results than SKO in terms of the final target algorithm performance achieved. Consequently, the remainder of our study focuses on the mechanisms that underly SPO.

In Sect. 15.4, we investigate the effectiveness of various methods for determining the set of parameter settings used for building the initial parameter response model. Here we found that using more complex initial designs did not consistently lead to improvements over more naïve methods. More importantly, we also found that parameter response models built from log-transformed performance measurements tended to be substantially more accurate than those built from raw data (as used by SPO). In Sect. 15.5, we turn to the sequential experimental design procedure. We introduce a simple variation in SPO's intensification mechanism which led to significant and substantial performance improvements. Next, we consider two previous expected improvement criteria for selecting the next parameter setting to evaluate, and derive a new expected improvement criterion specifically for optimization based on predictive models trained on log-transformed data. These theoretical improvements, however, did not consistently correspond to improvements in observed algorithm performance. Nevertheless, we demonstrate that overall, our novel variant of SPO—dubbed SPO$^+$—achieved an improvement over the best previously known results on the SAPS parameter optimization benchmark.

Section 15.6 presents the second part of our study, in which we investigate a different perspective on optimizing the parameters of an algorithm. Specifically, we explore the interactive use of statistical tools in parameter optimization, which (like automated tuning) is also supported by the SPOT. Our approach is based on *response surface methodology* (RSM), which can be characterized as a collection of mathematical and statistical techniques that are useful for the modeling and analysis of problems in which a response of interest is influenced by several variables and the objective is to maximize or minimize this response (Montgomery 2001, Beielstein 2003). We show how to assess the relative importance of each parameter as well as interactions between parameters, and to manually refine the region of interest based on this knowledge. While the automated tuning option requires nearly no statistical knowledge, it can be computationally expensive. In contrast, the interactive option requires some knowledge about statistics (especially classical regression analysis), but can reduce computational costs and help the algorithm designer to learn which parameters deserve attention. This is particularly useful in cases where target algorithm evaluations are costly compared to the overall computational resources and time available for the parameter optimization process.

Finally, Sect. 15.7 concludes the chapter with a discussion of advantages and drawbacks of automated and interactive approaches and the identification of interesting directions for future work.

15.2 Target Algorithms and Experimental Setup

Our first target algorithm was the *covariance matrix adaptation evolution strategy* (CMA-ES). CMA-ES is a prominent gradient-free global optimization algorithm for continuous functions (Hansen and Ostermeier 1996, Hansen and Kern 2004). It is based on an evolutionary strategy that uses a covariance matrix adaptation scheme. We used the Matlab implementation CMA-ES 2.54,[1] which is integrated into the SPO toolbox version 0.4 and was used as an example application for parameter optimization in the SPOT manual (Bartz-Beielstein et al. 2008a). CMA-ES has two obvious parameters: the number of parents, NPARENTS, and a factor NU ≥ 1 relating the number of parents to the population size. (The population size is defined as $\lfloor \text{NPARENTS} \times \text{NU} + 0.5 \rfloor$.) Bartz-Beielstein et al. (2008a) modified CMA-ES's interface to expose two additional parameters: the "learning rate for the cumulation for the step size control," CS, and the damping parameter, DAMPS (for details, see Hansen (2006), where NPARENTS, NU, CS, and DAMPS are called N, ν, c_σ, and d_σ, respectively). We used exactly the same *region of interest* (ROI; also called *experimental region*) considered in Bartz-Beielstein et al. (2008a)'s SPOT example based on CMA-ES. Table 15.1 provides a summary of the target algorithms, parameters, and regions of interest we used.

Table 15.1: Target algorithms, parameters, and the regions of interest (parameter domains) considered

Target algorithm	Parameter	Domain	Type
CMA-ES	NPARENTS	[1, 50]	integer
	NU	[2, 10]	continuous
	CS	(0, 1]	continuous
	DAMPS	[0.25, 0.99]	continuous
SAPS	α	(1, 1.4]	continuous
	ρ	[0, 1]	continuous
	P_{smooth}	[0, 0.2]	continuous
	wp	[0, 0.06]	continuous

For each run of CMA-ES, we allowed a limited number of function evaluations and used the resulting solution quality (i.e., the minimal function value found) as the response variable to be optimized. We considered four canonical 10-dimensional

[1] The newest CMA-ES version, 3.0, differs mostly in its interface and its support of "separable" CMA (see the change log at `http://www.lri.fr/~hansen/cmaes_inmatlab.html`).

test functions with a global minimum function value of zero that were previously used in published evaluations of CMA-ES. Specifically, we considered the Sphere function (used in the SPOT example mentioned above) and the Ackley, Griewangk, and Rastrigin functions (used by Hansen and Kern (2004)). Following Bartz-Beielstein et al. (2008a), for the Sphere function we initialized CMA-ES at the point $[10, ..., 10]^T \in \mathbb{R}^{10}$. To test global search performance, in the other test functions we initialized CMA-ES further away from the optima, at the point $[20, ..., 20]^T \in \mathbb{R}^{10}$. For the first two functions, we optimized mean solution quality reached by CMA-ES within $1,000$ function evaluations. For the latter two functions, which are more challenging, we set a limit of $10,000$ function evaluations. This setup is summarized in Table 15.2.

Table 15.2: Experimental setup for the CMA-ES test cases

Test function	Dimensionality	Initial point $[\cdot \mathbf{1} \in \mathbb{R}^{10}]$	# Function evaluations allowed
Sphere	10	10	$1,000$
Ackley	10	20	$1,000$
Griewangk	10	20	$10,000$
Rastrigin	10	20	$10,000$

The second target algorithm we considered was *Scaling And Probabilistic Smoothing* (SAPS) (Hutter et al. 2002), a high-performance dynamic local search algorithm for the propositional satisfiability problem. We used the standard UBCSAT implementation (Tompkins and Hoos 2004) of SAPS and defined the region of interest (Table 15.1) to closely follow an earlier parameter optimization study of SAPS by Hutter et al. (2007), with the difference that we did not discretize parameter values. (Hutter et al. did so because the parameter optimization procedure used in that work required it.) For SAPS, our goal was to minimize median runtime (measured in local search steps) for solving the "quasigroups with holes" (QWH) SAT instance used by Hutter et al. (2007). This instance belongs to a family of distributions that has received considerable interest in the SAT community. We chose this particular instance to facilitate direct comparison of the performance achieved by the parameter optimization procedures considered here and by Hutter et al. (2007).

To evaluate the quality $Q(\boldsymbol{\theta})$ of a proposed parameter setting $\boldsymbol{\theta}$ in an offline evaluation stage of the algorithm, we always performed additional test runs of the target algorithm with setting $\boldsymbol{\theta}$. In particular, for the CMA-ES test cases, we computed $Q(\boldsymbol{\theta})$ as the mean solution cost achieved by CMA-ES using $\boldsymbol{\theta}$ across 100 test runs. For the higher-variance SAPS domain, we computed $Q(\boldsymbol{\theta})$ as the median runtime achieved by SAPS with setting $\boldsymbol{\theta}$ across $1,000$ test runs. We define the *solution cost* c_k *of a parameter optimization run* after k runs of the target algorithm as the quality $Q(\boldsymbol{\theta})$ of the parameter setting $\boldsymbol{\theta}$ the method would output if terminated at that point. The *final performance* of a parameter optimization run is the performance at the end of the run. In order to measure the performance of a parameter optimization method, we performed 25 runs of the method with different random seeds, reporting mean and standard deviation of the final performance across these 25 repetitions.

Table 15.3: Summary of notation. The upper part of the table contains symbols used in the pseudocode, the middle part contains SPO parameters, and the lower part general notation

Symbol	Meaning
$\boldsymbol{\Theta}$	Space of allowable parameter settings (region of interest)
$\boldsymbol{\theta}$	Parameter setting, element of $\boldsymbol{\Theta}$
$\boldsymbol{\theta}_{i:j}$	Vector of parameter settings, $[\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1}, \ldots, \boldsymbol{\theta}_j]$
D	Dimensionality of $\boldsymbol{\Theta}$ (number of parameters to be tuned)
y	Response variable (performance of target algorithm)
history	Structure keeping track of target runs executed and responses, as well as incumbents
$N(\boldsymbol{\theta})$	Number of previous target algorithm runs with setting $\boldsymbol{\theta}$; depends on history
$\hat{c}(\boldsymbol{\theta})$	Empirical cost statistic over the $N(\boldsymbol{\theta})$ runs with $\boldsymbol{\theta}$ (e.g., mean runtime); depends on history
$\mathcal{M}$	Predictive model
r	Number of repeats in SPO (increases over time). Initial value: SPO parameter
$maxR$	Maximal number of repeats in SPO
d	Size of initial design in SPO
m	Number of parameter settings to evaluate in each iteration in SPO
p	Number of previous parameter setting to evaluate in each iteration of SPO$^+$
$Q(\boldsymbol{\theta})$	Test quality (cost) of parameter setting $\boldsymbol{\theta}$
c_k	Solution cost $Q(\boldsymbol{\theta})$ of incumbent parameter setting $\boldsymbol{\theta}$ at step k

We also performed paired Max-Wilcoxon signed rank tests for differences in final performance. We chose a paired test because, using identical random seeds, the ith repetition of every parameter optimization method used the same initial design and response values. For the experiments in Sect. 15.5.3 this pairing did not apply, and consequently, we used the (unpaired) Mann–Whitney U test instead.

15.3 Existing Methods for Sequential Model-Based Optimization of Noisy Functions

In this section, we review two existing model-based optimization methods for noisy responses: the sequential Kriging optimization (SKO) algorithm by Huang et al. (2006), and the sequential parameter optimization (SPO) procedure by Bartz-Beielstein et al. (2004b, 2005).

15.3.1 General Gaussian Process Regression

In order to model the dependence of a response variable (in our case, the performance of a given target algorithm) on parameter settings, both SKO and SPO use a form of Gaussian stochastic process, a nonparametric regression method that predicts both the mean and variance of a response variable. We first give the basic equations for Gaussian process regression and then describe the differences in the ways this approach is used by SKO and SPO.

To apply Gaussian process regression, first we need to select a parameterized kernel function $k_\lambda : \boldsymbol{\Theta} \times \boldsymbol{\Theta} \to \mathbb{R}^+$ that quantifies the similarity between two parameter settings. We also need to set the variance σ^2 of Gaussian-distributed measurement noise. The predictive distribution of a zero-mean Gaussian stochastic process for response y_{n+1} at input $\boldsymbol{\theta}_{n+1}$ given training data $\mathcal{D} = \{(\boldsymbol{\theta}_1, y_1), \ldots, (\boldsymbol{\theta}_n, y_n)\}$, measurement noise with variance σ^2, and kernel function k is then

$$p(y_{n+1}|\boldsymbol{\theta}_{n+1}, \boldsymbol{\theta}_{1:n}, \mathbf{y}_{1:n}) = \mathcal{N}(y_{n+1}|\boldsymbol{k}_*^{\mathrm{T}}[\boldsymbol{K}+\sigma^2\boldsymbol{I}]^{-1}\mathbf{y}_{1:n}), k_{**} - \boldsymbol{k}_*^{\mathrm{T}}(\boldsymbol{K}+\sigma^2\mathbf{I})^{-1}, \tag{15.1}$$

where

$$\boldsymbol{K} = \begin{pmatrix} k(\boldsymbol{\theta}_1, \boldsymbol{\theta}_1) & \ldots & k(\boldsymbol{\theta}_1, \boldsymbol{\theta}_n) \\ & \ddots & \\ k(\boldsymbol{\theta}_n, \boldsymbol{\theta}_1) & \ldots & k(\boldsymbol{\theta}_n, \boldsymbol{\theta}_n) \end{pmatrix} \tag{15.2}$$

$$\boldsymbol{k}_* = (k(\boldsymbol{\theta}_1, \boldsymbol{\theta}_{n+1}), \ldots, k(\boldsymbol{\theta}_n, \boldsymbol{\theta}_{n+1})) \tag{15.3}$$

$$k_{**} = k(\boldsymbol{\theta}_{n+1}, \boldsymbol{\theta}_{n+1}) + \sigma^2, \tag{15.4}$$

$\mathbf{I}$ is the n-dimensional identity matrix, and $p(a|b) = \mathcal{N}(a|c,d)$ denotes that the conditional distribution of a given b is a Gaussian with mean b and variance c; see, e.g., Rasmussen and Williams (2006) for a derivation. A variety of kernel functions can be used, the most common of which is of the form

$$K(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) = \sum_{k=1}^{d} \exp(-\lambda_k(\boldsymbol{\theta}_{ik} - \boldsymbol{\theta}_{jk})^2), \tag{15.5}$$

where $\lambda_1, \ldots, \lambda_d$ are the kernel parameters. This kernel is most appropriate if the response is expected to vary smoothly in the input parameters $\boldsymbol{\theta}$. The kernel parameters and the observation noise variance σ^2 constitute the *hyper-parameters* $\boldsymbol{\phi}$, which are typically set by maximizing the *marginal likelihood* $p(\boldsymbol{y}_{\mathbf{1:N}})$ with a gradient-based optimizer. Using the chain rule, the gradient is

$$\frac{\partial \log p(\boldsymbol{y}_{\mathbf{1:N}})}{\partial \phi_j} = \frac{\partial \log p(\boldsymbol{y}_{\mathbf{1:N}})}{\partial (\boldsymbol{K} + \sigma^2 \boldsymbol{I})} \frac{\partial (\boldsymbol{K} + \sigma^2 \boldsymbol{I})}{\partial \phi_j}.$$

In *noise-free* Gaussian process models, the observation noise variance is fixed to $\sigma^2 = 0$.

Learning a Gaussian stochastic process model from data can be computationally expensive. Inverting the $n \times n$ matrix $[\boldsymbol{K}+\sigma^2\mathbf{I}]$ takes time $O(n^3)$, and has to be done in every step of the hyper-parameter optimization. Various approximations can be used to reduce this to $O(n^2)$; see, e.g., Quinonero-Candela et al. (2007). We refer to the process of optimizing hyper-parameters and computing the inverse as *fitting* the model. Subsequent predictions are cheaper than fitting the model, requiring matrix-vector multiplications and thus time $O(n^2)$. This is still substantially slower than

Algorithm Framework 15.1: Sequential model-based Optimization

```
Input     : Target algorithm $A$
Output    : Incumbent parameter setting $\boldsymbol{\theta}_{inc}$
[history, $\boldsymbol{\theta}_{inc}$] ← Initialize();
$\mathcal{M} \leftarrow \emptyset$;
repeat
    $[\mathcal{M}, \boldsymbol{\theta}_{inc}]$ ← FitModel($\mathcal{M}$, history, $\boldsymbol{\theta}_{inc}$);
    $\boldsymbol{\Theta}_{new}$ ← SelectNewParameterSettings($\mathcal{M}$, $\boldsymbol{\theta}_{inc}$, history);
    [history, $\boldsymbol{\theta}_{inc}$] ← Intensify($\boldsymbol{\Theta}_{new}$, $\boldsymbol{\theta}_{inc}$, $\mathcal{M}$, history);
until TerminationCriterion() ;
return $\boldsymbol{\theta}_{inc}$;
```

Procedure 15.2: ExecuteRuns(history, $\boldsymbol{\theta}$, numRuns)

```
for $i = 1, \ldots,$ numRuns do
    Execute target algorithm $A$ with parameter setting $\boldsymbol{\theta}$, store response in $y$;
    Append $\boldsymbol{\theta}$ to history.$\boldsymbol{\theta}$;
    Append $y$ to history.$y$;
end
return history
```

prediction with a parametric model, the time complexity of which is independent of n.

15.3.2 A Common Framework for Sequential Model-Based Optimization

In this section, we describe SKO and SPO in a unified framework, which is outlined in the form of pseudocode in Algorithm Framework 15.1. One common building block of all parameter-optimization algorithms is a procedure that executes the target algorithm with a given parameter setting and stores the response. This building block is described in our Procedure 15.2, ExecuteRuns. On the first invocation, history.$\boldsymbol{\theta}$ and history.y are initialized to be empty lists.

Table 15.3 summarizes our notation and defines some global variables used in SKO and SPO. Note in particular $N(\boldsymbol{\theta})$ and $\hat{c}(\boldsymbol{\theta})$; $N(\boldsymbol{\theta})$ denotes the number of runs we have so far executed with a parameter setting $\boldsymbol{\theta}$, and $\hat{c}(\boldsymbol{\theta})$ denotes the empirical performance across the $N(\boldsymbol{\theta})$ runs that have been performed for $\boldsymbol{\theta}$.

15.3.2.1 Initialization

We outline the initialization of SKO and SPO in Procedures 15.3 and 15.4, respectively. Procedure Initialize() is called as

Procedure 15.3: Initialize() in SKO
$\boldsymbol{\Theta}$ denotes the space of allowable parameter settings (the region of interest); D denotes the number of parameters to be tuned.

```
$k \leftarrow 10D$;
$\boldsymbol{\theta}_{1:k} \leftarrow$ LatinHypercubeDesign($\boldsymbol{\Theta}$, $k$);
for $i = 1, \ldots, k$ do
    history $\leftarrow$ ExecuteRuns(history, $\boldsymbol{\theta}_i$, 1);
end
$\boldsymbol{\theta}_{k+1:k+D} \leftarrow$ the $D$ settings $\boldsymbol{\theta}_j$ out of $\boldsymbol{\theta}_{1:k}$ with smallest $\hat{c}(\boldsymbol{\theta}_j)$;
for $i = 1, \ldots, D$ do
    history $\leftarrow$ ExecuteRuns(history, $\boldsymbol{\theta}_{k+i}$, 1);
end
$\boldsymbol{\theta}_{inc} \leftarrow$ random element of $\arg\min_{\boldsymbol{\theta} \in \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_d\}} \hat{c}(\boldsymbol{\theta})$;
return [history, $\boldsymbol{\theta}_{inc}$];
```

Procedure 15.4: Initialize() in SPO
$\boldsymbol{\Theta}$ denotes the space of allowable parameter settings (the region of interest). The number of LHD parameter settings, d, and the number of repetitions, r, are parameters of SPO

```
$\boldsymbol{\theta}_{1:d} \leftarrow$ LatinHypercubeDesign($\boldsymbol{\Theta}$, $d$);
for $i = 1, \ldots, d$ do
    history $\leftarrow$ ExecuteRuns(history, $\boldsymbol{\theta}_i$, $r$);
end
$\boldsymbol{\theta}_{inc} \leftarrow$ random element of $\arg\min_{\boldsymbol{\theta} \in \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_d\}} \hat{c}(\boldsymbol{\theta})$;
history.$\boldsymbol{\Theta}_{hist} \leftarrow \{\boldsymbol{\theta}_{inc}\}$;
return [history, $\boldsymbol{\theta}_{inc}$]
```

$$[\text{history}, \boldsymbol{\theta}_{inc}] = \text{Initialize}().$$

SKO starts with a Latin hypercube design of $10 \times D$ parameter settings, where D is the number of parameters to be optimized. It executes the target algorithm at these settings and then performs an additional run for the D settings with the lowest response. The incumbent $\boldsymbol{\theta}_{inc}$ is chosen as the setting with the lowest empirical cost $\hat{c}(\boldsymbol{\theta})$ out of these D settings. In SPO, d parameter settings are chosen with a Latin hypercube design, and the target algorithm is executed r times for each of them; d and r are algorithm parameters. In practice, the initial design size d is chosen as the minimum number of points required to fit a reasonably accurate model. (Here, we used $d = 250$ and $r = 2$ as discussed in Sect. 15.4.1; however, in the direct comparison to SKO, we used the initial design of SKO within SPO to limit the number of confounding factors in the comparison.) The incumbent $\boldsymbol{\theta}_{inc}$ is chosen as the parameter setting with the lowest empirical cost $\hat{c}(\boldsymbol{\theta})$.

15.3.2.2 Fit of Response Surface Model

Both SKO and SPO base their predictions on a combination of a linear model and a Gaussian process (GP) model fit on the residual errors of the linear model. How-

Procedure 15.5: FitModel($\mathcal{M}$, history, $\boldsymbol{\theta}_{inc}$) in SKO
SKO sets its incumbent $\boldsymbol{\theta}_{inc}$ based on the learnt model

```
if $\mathcal{M} == \emptyset$ or last hyper-parameter optimization occurred more than $n$ steps ago then
    Fit GP model $\mathcal{M}$ and hyper-parameters for data $\{(\text{history}.\boldsymbol{\theta}_i, \text{history}.y_i)\}_{i \in \{1,\ldots,n\}}$;
else
    Fit GP model $\mathcal{M}$ for data $\{(\text{history}.\boldsymbol{\theta}_i, \text{history}.y_i)\}_{i \in \{1,\ldots,n\}}$, re-using
    hyper-parameters saved in previous $\mathcal{M}$;
end
$\boldsymbol{\Theta}_{seen} \leftarrow \bigcup_{i=1}^{n} \{\text{history}.\boldsymbol{\theta}_i\}$;
for all $\boldsymbol{\theta} \in \boldsymbol{\Theta}_{seen}$ do
    $[\mu_{\boldsymbol{\theta}}, \sigma^2_{\boldsymbol{\theta}}] \leftarrow$ Predict($\mathcal{M}$, $\boldsymbol{\theta}$);
end
$\boldsymbol{\theta}_{inc} \leftarrow$ random element of $\arg\min_{\boldsymbol{\theta} \in \boldsymbol{\Theta}}(\mu_{\boldsymbol{\theta}} + \sigma_{\boldsymbol{\theta}})$;
return $[\mathcal{M}, \boldsymbol{\theta}_{inc}]$;
```

Procedure 15.6: FitModel($\mathcal{M}$, history, $\boldsymbol{\theta}_{inc}$) in SPO
Recall that $\hat{c}(\boldsymbol{\theta})$ is the cost statistic across *all* runs with setting $\boldsymbol{\theta}$. SPO does not update its incumbent $\boldsymbol{\theta}_{inc}$ in this procedure

```
$\boldsymbol{\Theta}_{seen} \leftarrow \bigcup_{i=1}^{n} \{\text{history}.\boldsymbol{\theta}_i\}$;
Fit GP model $\mathcal{M}$ and hyper-parameters for data $\{\boldsymbol{\theta}, \hat{c}(\boldsymbol{\theta})\}_{\boldsymbol{\theta} \in \boldsymbol{\Theta}_{seen}}$, using fixed $\sigma^2 = 0$;
return $[\mathcal{M}, \boldsymbol{\theta}_{inc}]$;
```

ever, both of them default to using only a single constant basis function in the linear model, thus reducing the linear model component to a constant offset term, the mean response value. Throughout this chapter, we use these defaults; the model we use is thus an offset term plus a zero-mean GP model. Our implementation of SPO uses the DACE Matlab toolbox to construct this predictive model, while SKO implements the respective equations itself. The exact equations used in both SKO and the DACE toolbox implement methods to deal with ill-conditioning; we refer the reader to the original publications for details (Huang et al. 2006, Bartz-Beielstein 2006, Lophaven et al. 2002).

Procedure FitModel is called as

$$[\mathcal{M}, \boldsymbol{\theta}_{inc}] = \text{FitModel}(\mathcal{M}, \text{history}, \boldsymbol{\theta}_{inc}). \tag{15.6}$$

When it is first called, $\mathcal{M} = \emptyset$. Note that FitModel may update the incumbent configuration, θ_{inc}. SKO makes use of this, while SPO does not. (Instead, SPO updates θ_{inc} in Procedure Intensify.)

SKO uses Gaussian process regression in the conventional manner to fit noisy response data directly; we describe this in Procedure 15.5. Note that when a GP model is trained directly on noisy response data, measurement noise is assumed to be normally distributed, an assumption that is violated in many applications of parameter optimization. While distributions of solution qualities across multiple runs of a randomized heuristic algorithm can often be approximated quite well with a Gaussian distribution, it is well known that the distributions of runtimes of randomized heuristic algorithms for solving hard combinatorial problems tend to exhibit substantially fatter tails; see, e.g., Chap. 4 of Hoos and Stützle (2005). Also note that Gaussian process regression assumes symmetric noise and fits the *arithmetic mean* of the response that is used as input. Thus, if the inputs are raw response values, Gaussian processes model mean response. Commonly, the response value is transformed in order to yield a better model fit (see Sect. 15.4.2 for a more detailed discussion of transformations). In particular, a log transformation appears to be reasonable in many circumstances. However, note that under a log transformation, Gaussian process regression fits the mean of the *transformed* response, which corresponds to the geometric mean rather than the arithmetic mean of the true response.

As also described in Procedure 15.9, after fitting its model, SKO updates the new configuration, θ_{inc}, based on the new model. The parameter setting that minimizes a GP model's mean prediction is not necessarily the best choice for the incumbent, because it may be based on dramatically fewer function evaluations than other, similar-scoring configurations. Recognizing this fact, SKO implements a risk-averse strategy: it picks the previously evaluated parameter setting that minimizes predicted mean plus one predicted standard deviation.

SPO uses Gaussian process regression in a very different, nonstandard way, described in Procedure 15.6. It first computes the user-defined empirical performance metric $\hat{c}(\boldsymbol{\theta})$ for each parameter setting $\boldsymbol{\theta}$ evaluated so far, and then fits a *noise-free* GP model to learn a mapping from parameter settings to the performance metric. This approach has several benefits and drawbacks. If we have executed a single run with parameter setting $\boldsymbol{\theta}_1$ and many runs with setting $\boldsymbol{\theta}_2$, naturally our confidence in performance estimate $\hat{c}(\boldsymbol{\theta}_1)$ will be lower than our confidence in $\hat{c}(\boldsymbol{\theta}_2)$. SPO ignores this fact and collapses all information for a parameter setting $\boldsymbol{\theta}$ to a single value $\hat{c}(\boldsymbol{\theta})$, discarding all information on variance. On the other hand, fitting the GP model to the performance metric directly enables SPO to optimize almost arbitrary user-defined performance metrics, which could not be done with standard GP models. Examples include median performance, variance across runs, and trade-offs between mean and variance. To our best knowledge, SPO is the only existing model-based method with such flexibility in the objective function being optimized. Another benefit is that the assumption of normally-distributed response values is dropped. The final benefit of collapsing the data to a single point per parameter setting lies in the reduction in computational complexity thus achieved. While SKO has cubic scaling behavior in the number of target algorithm runs performed, SPO only takes time cubic in the number of *disjoint* parameter settings evaluated.

15.3.2.3 Selection of New Parameter Settings to Evaluate

Following Jones et al. (1998), both SKO and SPO use an expected improvement criterion (EIC) to determine which parameter settings to investigate next, thereby drawing on both the mean and variance predictions of the GP model. This criterion trades off learning about new, unknown parts of the parameter space and intensifying the search locally in the best known region (a so-called exploration/exploitation trade-off).

Procedure 15.7: SelectNewParameterSettings($\mathcal{M}, \boldsymbol{\theta}_{inc}$, history) in SKO

```
    $\boldsymbol{\Theta}_{new} \leftarrow$ the single parameter setting found by optimizing the augmented expected
    improvement criterion from (Huang et al. 2006) using the Nelder–Mead simplex method.
    return $\boldsymbol{\Theta}_{new}$
```

Procedure 15.8: SelectNewParameterSettings($\mathcal{M}, \boldsymbol{\theta}_{inc}$, history) in SPO
Note that m is a parameter of SPO

```
    // ===== Select $m$ parameter settings with expected improvement
    $\boldsymbol{\Theta}_{rand} \leftarrow$ set of $10,000$ elements drawn uniformly at random from $\boldsymbol{\Theta}$;
    for all $\boldsymbol{\theta} \in \boldsymbol{\Theta}_{rand}$ do
        $[\mu_{\boldsymbol{\theta}}, \sigma^2_{\boldsymbol{\theta}}] \leftarrow$ Predict($\mathcal{M}$, $\boldsymbol{\theta}$);
        $EI(\boldsymbol{\theta}) \leftarrow$ Compute expected improvement criterion $E[I^2(\boldsymbol{\theta})]$ (see Sect. 15.5.2)
        given $\mu_{\boldsymbol{\theta}}$ and $\sigma^2_{\boldsymbol{\theta}}$;
    end
    $\boldsymbol{\Theta}_{new} \leftarrow$ the $m$ elements of $\boldsymbol{\Theta}_{rand}$ with highest $EI(\boldsymbol{\theta})$;
    return $\boldsymbol{\Theta}_{new}$;
```

Procedure SelectNewParameterSettings is called as

$$\boldsymbol{\Theta}_{new} = \text{SelectNewParameterSettings}(\mathcal{M}, \boldsymbol{\theta}_{inc}, \text{history}). \tag{15.7}$$

SKO selects a single new parameter setting by maximizing an augmented expected improvement criterion using the Nelder-Mead simplex method. The augmentation adds a bias away from parameter settings for which predictive variance is low; see Huang et al. (2006). SPO, on the other hand, evaluates the $E[I^2]$ expected improvement criterion (Schonlau et al. 1998) at $10,000$ randomly-selected parameter settings, and chooses the m with the highest expected improvement; see Sect. 15.5.2 for more details. In this work, we use the default $m = 1$. For completeness, we give the simple pseudocode for these methods in Procedures 15.8 and 15.7.

15.3.2.4 Intensification

Any parameter optimization method must make decisions about which parameter setting $\boldsymbol{\theta}_{inc}$ to return to the user as its incumbent solution, both if interrupted during the search progress and (especially) upon completion. Candidate parameter settings are suggested by Procedure SelectNewParameterSettings, and in order to decide whether they, the current incumbent, or even another parameter setting should become the new incumbent, we need to perform additional runs of the target algorithm. Which parameter settings to use, how many runs to execute with each of them, and how to determine the new incumbent based on those runs is specified in Procedure Intensify, which is called as

$$[\text{history}, \boldsymbol{\theta}_{inc}] = \text{Intensify}(\boldsymbol{\Theta}_{new}, \boldsymbol{\theta}_{inc}, \mathcal{M}, \text{history}). \tag{15.8}$$

Procedure 15.9: Intensify($\boldsymbol{\Theta}_{new}$, $\boldsymbol{\theta}_{inc}$, $\mathcal{M}$, history) in SKO
SKO does not update its incumbent in this procedure

```
$\boldsymbol{\theta} \leftarrow$ the single element of $\boldsymbol{\Theta}_{new}$;
history $\leftarrow$ ExecuteRuns(history, $\boldsymbol{\theta}$, 1);
**return** [history, $\boldsymbol{\theta}_{inc}$];
```

Procedure 15.10: Intensify($\boldsymbol{\Theta}_{new}$, $\boldsymbol{\theta}_{inc}$, $\mathcal{M}$, history) in SPO 0.3
After performing runs for the incumbent and the new parameter settings, SPO updates the incumbent. Side effect: may increase the global number of repeats, r

```
**for all** $\boldsymbol{\theta} \in \boldsymbol{\Theta}_{new}$ **do**
    history $\leftarrow$ ExecuteRuns(history, $\boldsymbol{\theta}$, $r$);
**end**
history $\leftarrow$ ExecuteRuns(history, $\boldsymbol{\theta}_{inc}$, $r/2$);
$\boldsymbol{\Theta}_{seen} \leftarrow \bigcup_{i=1}^{n}\{\text{history}.\boldsymbol{\theta}_i\}$;
$\boldsymbol{\theta}_{inc} \leftarrow$ random element of $\arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}_{seen}} \hat{c}(\boldsymbol{\theta})$;
**if** $\boldsymbol{\theta}_{inc} \in$ history.$\boldsymbol{\Theta}_{hist}$ **then** $r \leftarrow \min\{2r, \text{maxR}\}$;
history.$\boldsymbol{\Theta}_{hist} \leftarrow$ history.$\boldsymbol{\Theta}_{hist} \cup \{\boldsymbol{\theta}_{inc}\}$;
**return** [history, $\boldsymbol{\theta}_{inc}$];
```

Note that this procedure allows an update of the incumbent, $\boldsymbol{\theta}_{inc}$. SPO makes use of this option, while SKO updates its incumbent in Procedure FitModel (see Procedure 15.5).

In order to provide more confident estimates for its incumbent, SPO implements an explicit intensification strategy. In SPO, predictive uncertainty cannot be used in the context of updating the incumbent parameter setting, because the underlying noise-free GP model predicts uncertainty to be exactly zero at all previously evaluated parameter settings. Instead, the number of evaluations performed for a parameter setting is used as a measure of confidence. SPO performs additional runs for its incumbent parameter setting $\boldsymbol{\theta}_{inc}$ in order to challenge that configuration. This is done to make sure that $\boldsymbol{\theta}_{inc}$ did not merely happen to yield low response values in the limited number of target algorithm runs previously performed with $\boldsymbol{\theta}_{inc}$. The number of evaluations used in this context differs between SPO versions 0.3 and 0.4; Procedures 15.10 and 15.11 reflect the differences between these two versions. In contrast, SKO does not implement an explicit intensification strategy; it only performs a single run with each parameter setting considered.

15.3.3 Empirical Comparison of SKO and SPO

We empirically compared SKO and SPO on the CMA-ES test cases, based on the same initial design (the one used by SKO) and with a limit of 200 runs of the target algorithm.

We chose this low limit on the number of runs because the original SKO implementation was very slow: even limited to as few as 200 runs of the target algorithm,

Procedure 15.11: Intensify($\boldsymbol{\Theta}_{new}$, $\boldsymbol{\theta}_{inc}$, $\mathcal{M}$, history) in SPO 0.4
After performing runs for the incumbent and the new parameter settings, SPO updates the incumbent. Side effect: may increase the global number of repeats, r

```
for all $\boldsymbol{\theta} \in \boldsymbol{\Theta}_{new}$ do
    history $\leftarrow$ ExecuteRuns(history, $\boldsymbol{\theta}$, $r$);
end
history $\leftarrow$ ExecuteRuns(history, $\boldsymbol{\theta}_{inc}$, $r - N(\boldsymbol{\theta}_{inc})$);
$\boldsymbol{\Theta}_{seen} \leftarrow \bigcup_{i=1}^{n} \{\text{history}.\boldsymbol{\theta}_i\}$;
$\boldsymbol{\theta}_{inc} \leftarrow$ random element of $\arg\min_{\boldsymbol{\theta} \in \boldsymbol{\Theta}_{seen}} \hat{c}(\boldsymbol{\theta})$;
if $\boldsymbol{\theta}_{inc} \in \text{history}.\boldsymbol{\Theta}_{hist}$ then $r \leftarrow \min\{r+1, \text{maxR}\}$;
history.$\boldsymbol{\Theta}_{hist} \leftarrow$ history.$\boldsymbol{\Theta}_{hist} \cup \{\boldsymbol{\theta}_{inc}\}$;
return [history, $\boldsymbol{\theta}_{inc}$];
```

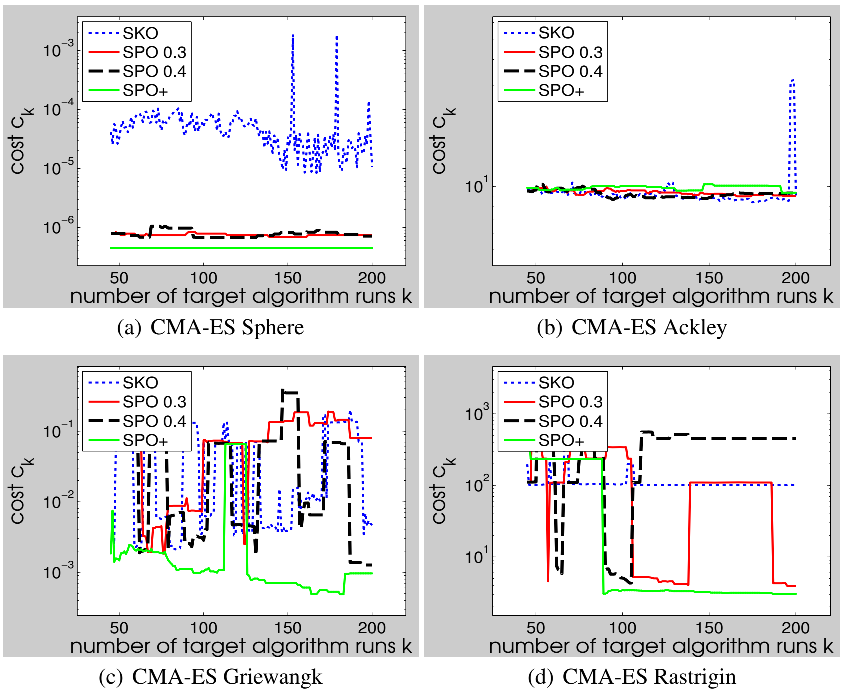
SKO runs took about one hour.[2] SKO spent most of its time performing numerical optimization of the expected improvement criterion. The iterations of SKO became slower as a run progressed, and therefore we could only afford to perform tuning runs for 200 runs of the target algorithm. For this reason and because SKO can only optimize *mean* performance, we did not perform experiments with SKO for the SAPS scenario, which features quite high observation noise and in which the objective is to minimize *median* runtime.

We reimplemented SPO 0.3 and 0.4, as well as a new version, SPO$^+$ (defined in Sect. 15.5.1). We verified that the performance of our reimplemented SPO 0.4 matched that of the original SPO 0.4 implementation. The SPO runs were substantially faster than those of SKO. They took about 2 minutes per repetition,[3] 85% of which was spent running the target algorithm.

Our first set of experiments used original, untransformed CMA-ES solution quality as the objective function to be minimized; we show the results in Fig. 15.1. On the Sphere function, the LHD already contained very good parameter settings, and the challenge was mostly to select the best of these and stick with it. From the figure, we observe that SPO largely succeeded in doing this, while SKO did not. On the Ackley function, SKO's performance was quite good, except for a drastic spike close to 200 runs of the target algorithm. On the Griewangk and Rastrigin functions, the variation of performance across multiple runs of CMA-ES was very high. Cor-

[2] We ran SKO on a 3 GHz Pentium 4 with 4 GB RAM running Windows XP Professional, Service Pack 3. We report wall clock time on an otherwise idle system. (We did not use Linux machines for these experiments because SKO only compiled for Windows.) In order to ascertain that the target algorithm has exactly the same behavior as for other methods running under Linux, we gathered the function values SKO requested by means of a wrapper script that connected to the machines the SPO experiments were carried out on, performed a requested run of the target algorithm there, and returned the result of the run. This incurred very little overhead. Finally, some SKO runs failed due to problems in the numerical optimization of the expected improvement criterion. We repeated those runs until they completed. These repetitions were nondeterministic due to measurement noise in the objective function.

[3] These experiments were carried out on a cluster of 55 dual 3.2 GHz Intel Xeon CPUs with 2 GB RAM each and 2 MB cache per CPU, running OpenSuSE Linux 10.1. Each run only used one CPU and we report CPU time.



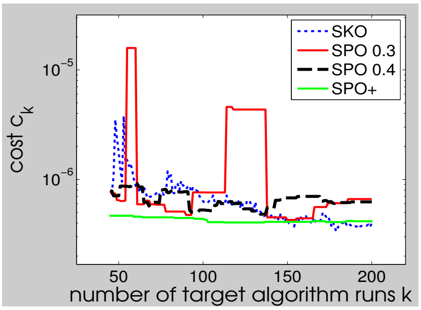
(a) CMA-ES Sphere (b) CMA-ES Ackley

(c) CMA-ES Griewangk (d) CMA-ES Rastrigin

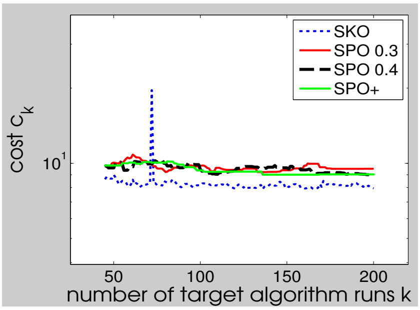
Fig. 15.1: Comparison of SKO and three variants of SPO for optimizing CMA-ES on the Sphere function. We plot the solution cost c_k of each method (mean solution quality CMA-ES achieved in 100 test runs on each of the 4 test functions using the method's chosen parameter settings) as a function of the number of algorithm runs, k, it was allowed to perform. These values are averaged across 25 runs of each method. All models were based on SKO's initial design and original untransformed data

respondingly, all approaches showed large variation in the quality of the parameter settings they selected over time. (Intuitively, the parameter optimization procedure detects a new region, which seems promising based on a few runs of the target algorithm. Then, after additional target algorithm runs, that region is discovered to be worse than initially thought, and the optimizer moves on. During the period it takes to discover the true, poor nature of the region, the search returns a poor incumbent.) SPO$^+$ showed the most robust performance of the four parameter optimization procedures.

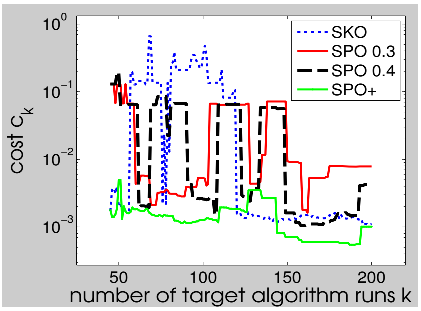
Secondly, we experimented with log-transformed qualities (see Sect. 15.4.2 for a more detailed discussion of log-transformations). Figure 15.2 shows that this transformation improved SKO performance quite substantially and did not affect the SPO variants much. We attribute this to the fact that the quality of the model is much more important in SKO. While SPO "only" uses the model in order to make decisions about the next parameter setting to evaluate, SKO also uses it in order to select its incumbent. After the log transformation, SKO and SPO$^+$ performed comparably.



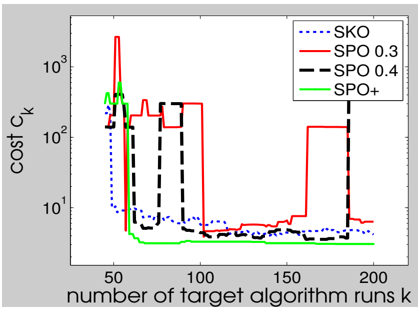
(a) CMA-ES Sphere



(b) CMA-ES Ackley



(c) CMA-ES Griewangk



(d) CMA-ES Rastrigin

Fig. 15.2: Comparison of SKO and three variants of SPO for optimizing CMA-ES on the Sphere function. We plot the solution cost c_k of each method (mean solution quality CMA-ES achieved in 100 test runs on each of the 4 test functions using the method's chosen parameter settings) as a function of the number of algorithm runs, k, it was allowed to perform. These values are averaged across 25 runs of each method. All models were based on SKO's initial design and log-transformed data (for SPO as discussed in Sect. 15.4.2)

For three main reasons, we decided to focus the remainder of this study on various aspects of the SPO framework. Firstly, our main interest is in complex scenarios, in which the predictive model might not actually perform very well. In such scenarios, we believe it is important to employ an explicit intensification criterion instead of relying on the model alone to select incumbents. Secondly, SPO has the advantage of being able to optimize a variety of user-defined objective functions, while the Gaussian process model underlying SKO can only fit the mean of the data. In practice, users might be more interested in statistics other than mean performance, such as the best out of ten algorithm runs. SPO implements this performance metric and also allows for many other options. Finally, the SKO implementation we used was simply too slow to perform the type of experiments with tens of the thousands of target algorithm runs that interested us. Nevertheless, a worthwhile direction for future work would be to experiment with explicit intensification mechanisms in the SKO framework.

15.4 Model Quality

It is not obvious that a model-based parameter optimization procedure needs models that accurately predict target algorithm performance across all parameter settings, particularly including very bad ones. Nevertheless, all else being equal, models with good overall accuracy are generally helpful to such methods, and are furthermore essential to more general tasks such as performance robustness analysis. In this section, we investigate the effects of two key model-design choices on the accuracy of the GP models used by SPO.

15.4.1 Choosing the Initial Design

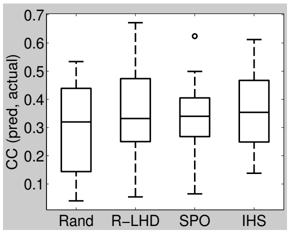
In the overall approach described in Algorithm Framework 15.1, an initial parameter response model is determined by constructing a GP model based on the target algorithm's performance on a given set of parameter settings (the *initial design*). This initial model is then subsequently updated based on runs of the target algorithm ith additional parameter settings. The decision about which additional parameter settings to select is based on the current model.

It is reasonable to expect that the quality of the final model (and the performance-optimizing parameter setting determined from it) would depend on the quality of the initial model. Therefore, we studied the overall accuracy of the initial parameter response models constructed based on various initial designs. The effect of the number of parameter settings in the initial design, d, as well as the number of repetitions for each parameter setting, r, has been studied before (Bartz-Beielstein and Preuss 2006), and we thus fixed them in this study.[4] Specifically, we used $r = 2$ and $d = 250$, such that when methods were allowed $1,000$ runs of the target algorithm, half of them were chosen with the initial design.

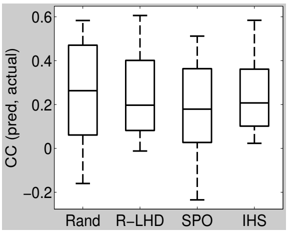
Here, we study the effect of the method for choosing *which* 250 parameter settings to include in the initial design, considering four methods: (1) a uniform random sample from the region of interest; (2) a random Latin hypercube design (LHD); (3) the LHD used in SPO; and (4) a more complex LHD based on *iterated distributed hypercube sampling* (IHS) (Beachkofski and Grandhi 2002).

We evaluated the parameter response models obtained using these initialisation strategies by assessing how closely model predictions at previously unseen parameter settings matched the true performance achieved using these settings. In particular, we were interested in how useful the predictions were for determining whether a given parameter setting performed better or worse than others. In order to answer this question, we evaluated the Spearman correlation coefficient between true and predicted performance for 250 randomly-sampled test parameter settings. This

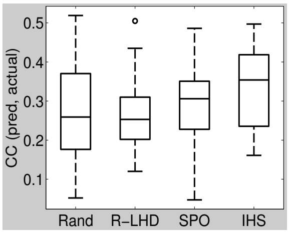
[4] We do note, however, that this previous work has not conclusively answered the question of how best to set the initial design size, and thus that this question continues to present an open research problem.



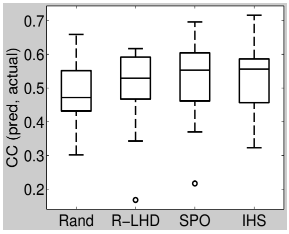
(a) CMAES-Sphere



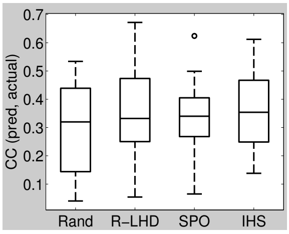
(b) CMAES-Ackley



(c) CMAES-Griewangk

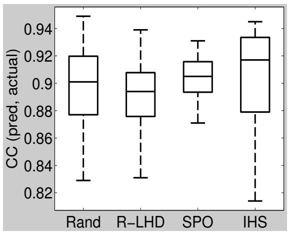


(d) CMAES-Rastrigin

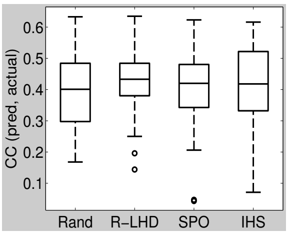


(e) SAPS-QWH

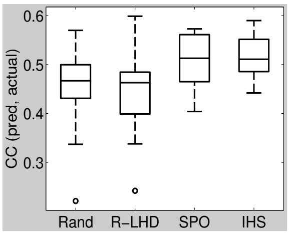
Fig. 15.3: Performance of models based on different initial designs and raw, untransformed data. We computed the Spearman correlation coefficient (CC) between actual and predicted performance for 250 randomly selected test parameter settings and show boxplots across 25 independent repetitions. Each repetition used the same test parameter settings



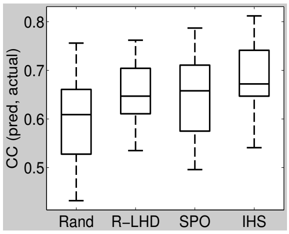
(a) CMAES-Sphere



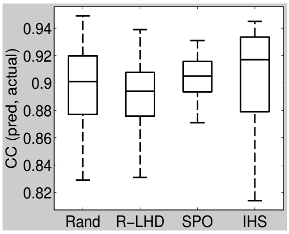
(b) CMAES-Ackley



(c) CMAES-Griewangk



(d) CMAES-Rastrigin



(e) SAPS-QWH

Fig. 15.4: Performance of models based on different initial designs and log-transformed data. We compute the Spearman correlation coefficient (CC) between actual and predicted performance for 250 randomly selected test parameter settings and show boxplots across 25 independent repetitions. Each repetition used the same test parameter settings

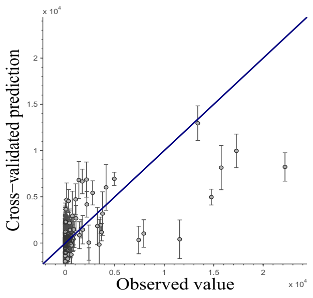
value lies on the interval [-1,1], with 1 indicating perfect correlation of the predicted and the true ranks, 0 indicating no correlation, and -1 perfect anti-correlation.

The results of this analysis for our five test cases are summarized in Fig. 15.3. Overall, for the original untransformed data we observed little variation in predictive quality due to the procedure used for constructing the initial design. This observation is consistent with others that have been made in the literature; for example, Santner et al. (2003, p.149) state: "It has not been demonstrated that LHDs are superior to any designs other than simple random sampling (and they are only superior to simple random sampling in some cases)."

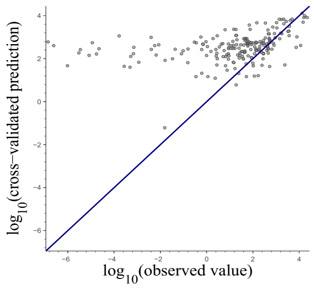
15.4.2 Transforming Performance Data

The second issue we investigated was whether more accurate models could be obtained by using log-transformed performance data for building and updating the model. Our consideration of this transformation was motivated by the fact that our main interest is in minimizing positive functions with spreads of several orders of magnitude that arise in the optimization of runtimes. Indeed, we have used log-transformations for predicting runtimes of algorithms in different contexts before (see., e.g., Leyton-Brown et al. (2002)). All of the test functions we consider for CMA-ES are also positive functions; in general, non-positive functions can be transformed to positive functions by subtracting a lower bound. In the context of model-based optimization, log transformations were previously advocated by Jones et al. (1998). The problem studied in that work was slightly different in that the functions considered were noise free. We adapt Jones et al. 's approach by first computing performance metrics (such as median runtime) and then fitting a GP model to the log-transformed metrics. Note that this is different from fitting a GP model to the log-transformed noisy data as done by Williams et al. (2000) and Huang et al. (2006). As discussed earlier, the performance metric that is implicitly optimized under this latter approach is geometric mean performance, which is a poor choice in situations where performance variations change considerably depending on the parameter setting. In contrast, when first computing performance metrics and then applying a transformation, any performance metric can be optimized, and we do not need to assume a Gaussian noise model. Finally, Bartz-Beielstein et al. (2008b) and Konen et al. (2009) report similar results based on square-root or cube-root transformations as indeed does Leyton-Brown (2003); see also the discussion of transformations in Chap. 2 of this book.

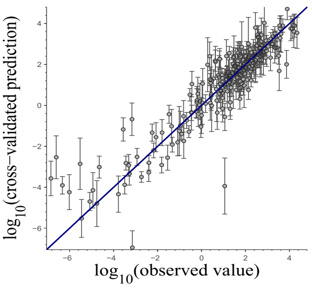
We experimentally evaluated the impact of log transformations. For the same data as in the previous section (initial designs with 250 parameter settings, 2 repetitions each), we fit noise-free GP models based on log-transformed cost statistics. That is, for each of the 250 parameter settings, $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{250}$, we first computed the mean of the two responses, $\hat{c}(\boldsymbol{\theta}_i)$, and then constructed a model using the data $\{(\boldsymbol{\theta}_1, \hat{c}(\boldsymbol{\theta}_1)), \ldots, (\boldsymbol{\theta}_{250}, \hat{c}(\boldsymbol{\theta}_{250}))\}$. We then computed predictions for the same 250 test parameters settings used in the previous section and evaluated the Spearman



(a) Untransformed data.



(b) Untransformed data on log-log scale; only means are plotted.



(c) Log-transformed data on log-log scale.

Fig. 15.5: Performance of noise-free GP models for CMA-ES-sphere based on an initial design using a random LHD; in (a) and (c) we plot mean ± one standard deviation of the prediction. For better visual comparison to (c), (b) shows exactly the same mean predictions as (a), but on a log–log scale, restricted to the 228/250 data points whose predicted response values were positive

correlations between predicted and true responses. As can be seen from the results reported in Fig. 15.4, the use of log-transformed performance data tended to result in much better model accuracy than the use of raw performance data. In some cases, the improvements were quite drastic. For example, for CMA-ES-sphere, the Spearman correlation coefficient improved from below 0.4 to above 0.9 when using models based on log-transformed performance data. Figure 15.5 illustrates the predictive accuracy and predictive uncertainty of these two models.

15.5 Sequential Experimental Design

Having studied the initial design, we now turn our attention to the sequential search for performance-optimizing parameters. Since log transformations consistently led to improved performance, and since random LHDs yielded performance comparable to that of more complex designs, we fixed these two design choices.

15.5.1 Intensification Mechanism

In order to achieve good results when optimizing parameters based on a noisy performance metric such as runtime or solution quality achieved by a randomized algorithm, it is important to perform a sufficient number of runs for the parameter settings considered. However, runs of a given target algorithm on interesting problem instances are typically computationally expensive. There is thus a delicate trade-off between the number of algorithm runs performed on each parameter setting and the number of parameter settings considered over the course of the optimization process.

Realizing the importance of this trade-off, SPO implements a mechanism for gradually increasing the number of runs to be performed for each parameter setting during the parameter optimization process. In particular, SPO increases the number of runs to be performed for each subsequent parameter setting whenever the incumbent $\boldsymbol{\theta}_{inc}$ selected in an iteration was already the incumbent in some previous iteration. SPO 0.3 (Bartz-Beielstein et al. 2005, Bartz-Beielstein 2006, Bartz-Beielstein and Preuss 2006) doubles the number of subsequent target algorithm runs whenever this happens; SPO 0.4 only increments the number of runs by one each time; see Procedures 15.10 and 15.11 in Sect. 15.3.2.4. Both versions perform additional runs for the current incumbent, $\boldsymbol{\theta}_{inc}$, to make sure it is used in as many runs of the target algorithm as any new parameter setting. [5]

While the intensification mechanisms of SPO 0.3 and SPO 0.4 work well in most cases, we have encountered runs of SPO in high-noise scenarios in which there are many parameter settings with few, "lucky" target algorithm runs that allow them to become incumbents. In those runs of SPO, a new incumbent was picked in almost every iteration, because the previous incumbent had been found to be poor after additional runs were performed on it. This situation continued throughout the entire parameter optimization process, leading to a final choice of parameter settings that had only been evaluated using very few ("lucky") target algorithm runs and that performed poorly in independent test runs.[6]

[5] Another approach for allocating an appropriate number of target algorithm runs to each parameter setting is Chen et al. (2003)'s *optimal computational budget allocation* (OCBA). Lasarczyk (2007) implemented OCBA as an explicit intensification method to improve SPO's selection procedure, especially in high-noise scenarios. This implementation was done in R (Ihaka and Gentleman 1996), and forthcoming versions of SPOT, which will also be based on R, will include OCBA.

[6] One possible explanation of this scenario is that SPO has problems when the number of LHD points is large (e.g., $d = 250$ in our experiments) in high-noise scenarios. SPO selects its incumbent as the previously visited parameter setting with the best empirical performance across the runs performed with it. All parameter settings in the LHD count as "previously visited". Thus, the

Procedure 15.12: Intensify($\boldsymbol{\Theta}_{new}$, $\boldsymbol{\theta}_{inc}$, $\mathcal{M}$, history) *in SPO*$^+$
Recall that $N(\boldsymbol{\theta})$ denotes the number of algorithm runs which have been performed for $\boldsymbol{\theta}$; it depends on the history

```
for all $\boldsymbol{\theta} \in \boldsymbol{\Theta}_{new}$ do
    $r \leftarrow 1$;
    history ← ExecuteRuns(history, $\boldsymbol{\theta}$, 1);
    numBonus ← 1;
    if $N(\boldsymbol{\theta}) > N(\boldsymbol{\theta}_{inc})$ then
        history ← ExecuteRuns(history, $\boldsymbol{\theta}_{inc}$, 1);
        numBonus ← 0;
    end
    while true do
        if $\hat{c}(\boldsymbol{\theta}) > \hat{c}(\boldsymbol{\theta}_{inc})$ then
            // ===== Reject challenger, perform bonus runs for $\boldsymbol{\theta}_{inc}$
            history ← ExecuteRuns(history, $\boldsymbol{\theta}_{inc}$,
            $\min(\text{numBonus}, \text{maxN} - N(\boldsymbol{\theta}_{inc}))$);
            break;
        end
        if $N(\boldsymbol{\theta}) \geq N(\boldsymbol{\theta}_{inc})$ then
            // ===== Challenger becomes incumbent
            $\boldsymbol{\theta}_{inc} \leftarrow \boldsymbol{\theta}$;
            break;
        end
        if TerminationCriterion() then
            return [$\boldsymbol{\theta}_{inc}$, history];
        end
        $r \leftarrow \min(2r, N(\boldsymbol{\theta}_{inc}) - N(\boldsymbol{\theta}))$;
        history ← ExecuteRuns(history, $\boldsymbol{\theta}$, $r$);
        numBonus ← numBonus $+r$;
    end
end
return [history, $\boldsymbol{\theta}_{inc}$];
```

This observation motivated us to introduce a different intensification mechanism that guarantees increasing confidence about the performance of the parameter settings we select as incumbents. In particular, inspired by the mechanism used in FocusedILS (Hutter et al. 2007), we maintain the invariant that we never choose an incumbent unless it is the parameter setting with the most function evaluations. Promising parameter settings are given additional function evaluations until either they cease to appear promising or they receive enough function evaluations to become the new incumbent. We provide pseudocode for this new intensification mechanism in Procedure 15.12.

In detail, our new intensification mechanism works as follows. In the first iteration, the incumbent is chosen exactly as in SPO, because all parameter settings receive the same number of function evaluations. From then on, in each iteration we select a set of parameter settings and compare them to the incumbent $\boldsymbol{\theta}_{inc}$. We

larger the LHD, the more runs need to be performed in order for decisions about the incumbent to be based on a minimum number of runs.

Procedure 15.13: SelectNewParameterSettings($\mathcal{M}, \boldsymbol{\theta}_{inc}$, history) in *SPO*$^+$
Recall that p and m are parameters of SPO$^+$. We use $m = 1$ and $p = 5$ in our experiments

```
// ===== Select $m$ parameter settings with expected improvement
$\boldsymbol{\Theta}_{rand} \leftarrow$ set of $10,000$ elements drawn uniformly at random from $\boldsymbol{\Theta}$;
for all $\boldsymbol{\theta} \in \boldsymbol{\Theta}_{rand}$ do
    $[\mu_{\boldsymbol{\theta}}, \sigma^2_{\boldsymbol{\theta}}] \leftarrow$ Predict($\mathcal{M}$, $\boldsymbol{\theta}$);
    $EI(\boldsymbol{\theta}) \leftarrow$ Compute expected improvement criterion given $\mu_{\boldsymbol{\theta}}$ and $\sigma^2_{\boldsymbol{\theta}}$;
end
$\boldsymbol{\Theta}_{new} \leftarrow$ the $m$ elements $\boldsymbol{\theta}$ of $\boldsymbol{\Theta}_{rand}$ with highest $EI(\boldsymbol{\theta})$;
// ===== Select $p$ previously used parameter settings
$\boldsymbol{\Theta} \leftarrow \bigcup_{i=1}^{n} \{\text{history}.\boldsymbol{\theta}_i\}$;
$\boldsymbol{\Theta}_{previous} \leftarrow p$ elements $\boldsymbol{\theta} \in \boldsymbol{\Theta}$, drawn without repetitions with prob. proportional to $1/\hat{c}(\boldsymbol{\theta})$;
return $\boldsymbol{\Theta}_{new} \cup \boldsymbol{\Theta}_{previous}$;
```

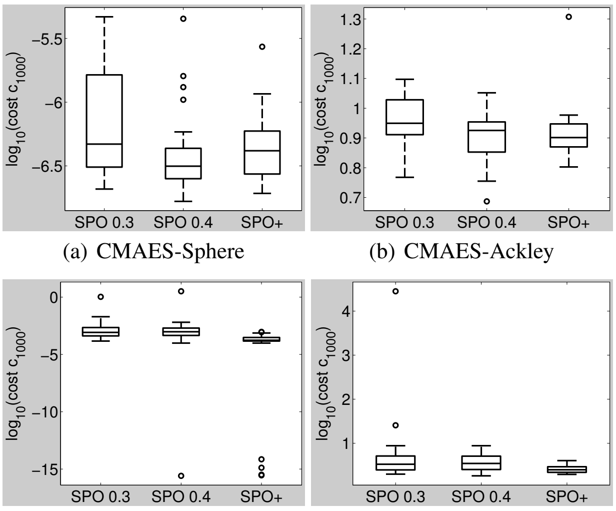
denote the number of runs that have so far been executed with parameter setting $\boldsymbol{\theta}$ as $N(\boldsymbol{\theta})$, and the corresponding empirical performance as $\hat{c}(\boldsymbol{\theta})$. For each selected setting $\boldsymbol{\theta}$, we iteratively perform runs until $N(\boldsymbol{\theta}) \geq N(\boldsymbol{\theta}_{inc})$ and/or $\hat{c}(\boldsymbol{\theta}) > \hat{c}(\boldsymbol{\theta}_{inc})$.[7] Whenever we reach a point where $N(\boldsymbol{\theta}) \geq N(\boldsymbol{\theta}_{inc})$ and $\hat{c}(\boldsymbol{\theta}) \leq \hat{c}(\boldsymbol{\theta}_{inc})$, we select $\boldsymbol{\theta}$ as the new incumbent. On the other hand, if we ever observe $\hat{c}(\boldsymbol{\theta}) > \hat{c}(\boldsymbol{\theta}_{inc})$, we reject $\boldsymbol{\theta}$. Note this criterion for rejection is very aggressive. Indeed, rejection frequently occurs after a single run, at a point where a statistical test would not be able to conclude that $\boldsymbol{\theta}$ is worse than $\boldsymbol{\theta}_{inc}$. Upon rejecting a configuration $\boldsymbol{\theta}$, we also perform as many additional runs for $\boldsymbol{\theta}_{inc}$ as were just performed for evaluating $\boldsymbol{\theta}$. This ensures that the number of runs used for intensification is comparable to that used for exploration of new parameter settings.

The parameter settings we evaluate against $\boldsymbol{\theta}_{inc}$ at each iteration include one new parameter setting selected based on an expected improvement criterion (here $E[I^2]$, see Sect. 15.5.2). They also include p previously evaluated parameter settings $\boldsymbol{\theta}_{1:p}$, where p is an algorithm parameter and in this work always set to 5. This set is constructed by selecting p previously evaluated settings $\boldsymbol{\theta}$ with probability proportional to $1/\hat{c}(\boldsymbol{\theta})$, without replacement. Procedure 15.13 provides pseudocode for this selection of parameter settings to evaluate against $\boldsymbol{\theta}_{inc}$.

This mechanism guarantees that at each step there will be a positive probability of reevaluating a potentially optimal parameter setting after it has been rejected. It allows us to be aggressive in rejecting new candidates, since we can always get back to the most promising ones. Note that if the other SPO variants (0.3 and 0.4) discover the true optimal parameter setting $\boldsymbol{\theta}_{best}$ but observe one or more very "unlucky" runs on it, $\boldsymbol{\theta}_{best}$ will never be revisited, because the underlying noise-free GP model attributes a high mean and zero uncertainty to any previously visited parameter setting for which poor empirical performance has been observed across the target

[7] We batch runs to reduce overhead, starting with a single new run for each $\boldsymbol{\theta}$ and doubling the number of new runs iteratively up to a maximum of $N(\boldsymbol{\theta}_{inc}) - N(\boldsymbol{\theta})$ runs.

algorithm runs performed for it. Therefore, no expected improvement criterion will select such a parameter setting again in later iterations.



(a) CMAES-Sphere (b) CMAES-Ackley

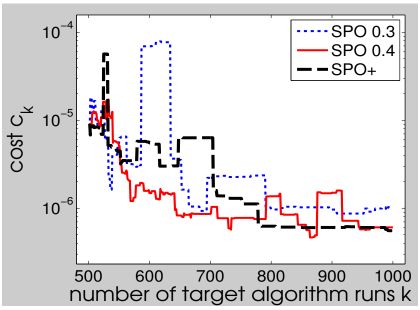
(c) CMAES-Griewangk (d) CMAES-Rastrigin

Fig. 15.6: Comparison of different intensification mechanisms for optimizing CMA-ES performance. We show boxplots of solution cost $c_{1,000}$ achieved in the 25 runs of each parameter optimizer for each test case

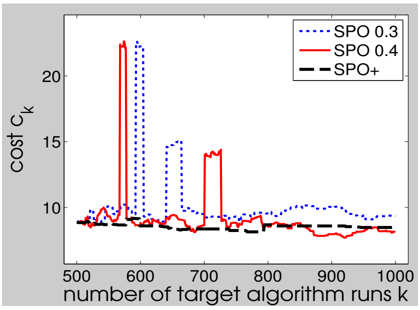
Table 15.4: The p-values for pairwise comparisons of different intensification mechanisms for optimizing the performance of CMA-ES on our standard benchmarks. These p-values correspond to the data in Fig. 15.6 and are based on a pairwise Max-Wilcoxon test as described in Sect. 15.2

	Sphere	Ackley	Griewangk	Rastrigin
SPO 0.3 versus SPO 0.4	0.07	**0.015**	0.95	1
SPO 0.3 versus SPO$^+$	0.20	**0.020**	**0.00006**	**0.0005**
SPO 0.4 versus SPO$^+$	0.56	0.97	**0.00009**	**0.0014**

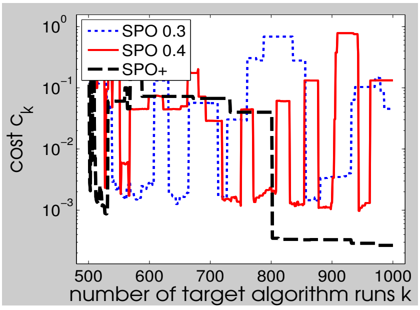
We denote as SPO$^+$ the variant of SPO that uses a random LHD, log-transformed data (for positive functions only; otherwise untransformed data), expected improvement criterion $E[I^2]$, and the new intensification criterion just described. We compared SPO 0.3, SPO 0.4, and SPO$^+$—all based on a random LHD and log-transformed data—for our CMA-ES test cases and summarize the results in Fig. 15.6 and Table 15.4. For the Ackley function, SPO 0.4 performed best on average, but only insignificantly better than SPO$^+$, one of whose runs performed quite poorly. For the Sphere function, on average SPO$^+$ performed insignificantly better than the other SPO variants, showing better median performance and no poor outliers among its 25 runs. For the other two functions, SPO$^+$ performed both significantly and substantially better than either SPO 0.3 or SPO 0.4, finding parameter settings that led to CMA-ES performance orders of magnitude better than those obtained from SPO 0.3.



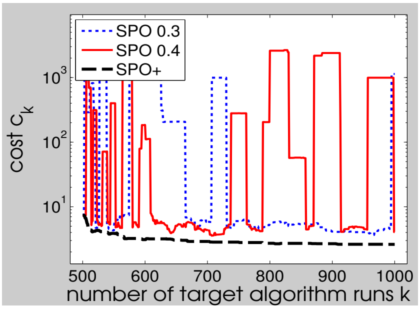
(a) CMA-ES Sphere



(b) CMA-ES Ackley



(c) CMA-ES Griewangk

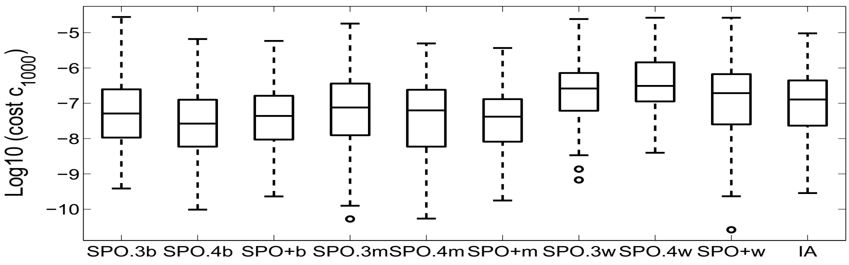


(d) CMA-ES Rastrigin

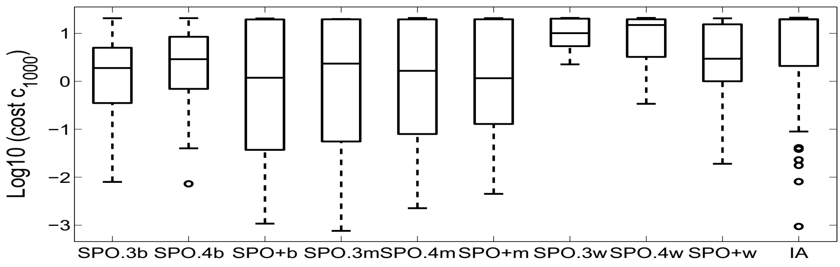
Fig. 15.7: Solution cost c_k (mean solution quality CMA-ES achieved in 100 test runs using the method's chosen parameter settings) of SPO 0.3, SPO 0.4, and SPO$^+$, as a function of the number of target algorithm runs, k, the method is allowed. We plot means of c_k across 25 repetitions of each parameter optimization procedure

More importantly, as can be seen in Fig. 15.7, over the course of the optimization process, SPO$^+$ showed much less variation in the quality of the incumbent parameter setting than the other SPO variants did. This was the case even for the Ackley function, where SPO$^+$ did not perform best on average at the very end of its trajectory, and can also be seen on the Griewangk and Rastrigin functions, where SPO$^+$ clearly produced the best results.

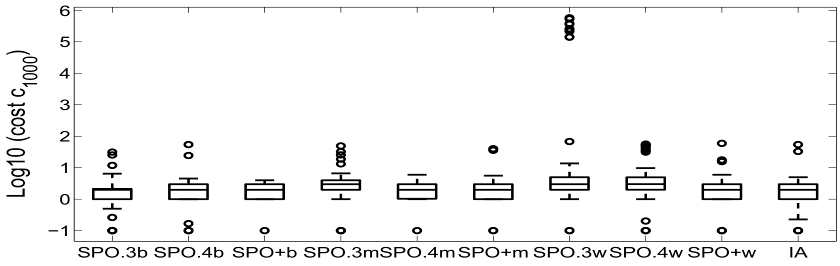
We now take a more in-depth look at how the mean solution costs c_k come about. To do this, for each of the four test cases, we extracted the finally-chosen parameter settings from nine automated parameter optimization runs: the best, median, and worst settings of each of SPO 0.3, SPO 0.4, and SPO$^+$, all with respect to test



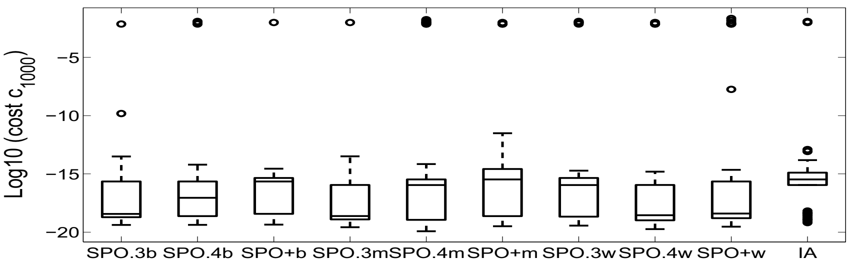
(a) CMAES-Sphere



(b) CMAES-Ackley

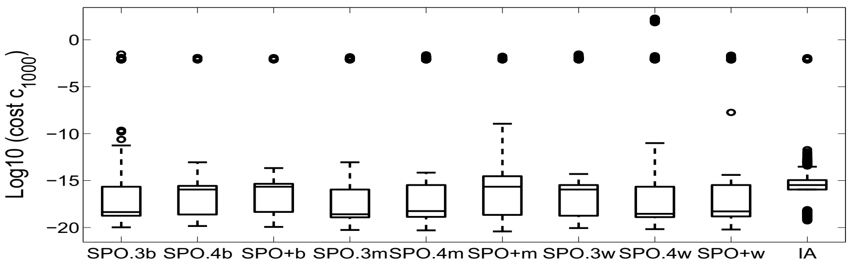


(c) CMAES-Rastrigin



(d) CMAES-Griewangk

Fig. 15.8: Boxplots comparing automatically-identified parameter settings to those found with the interactive procedure. In the axis labels, “b” stands for the parameter setting of the best run of that parameter optimizer (with respect to “original” test performance), “m” stands for the median-best one, and “w” for the worst. IA stands for the interactively found setting. Each run of the interactive process used fewer than 200 function evaluations, whereas the automated optimization procedures were allowed 1,000 evaluations per run. In order to avoid clutter in subfigure (c), we plot data points below 0.1 as 0.1 (-1 after $\log_{10}$ transformation); note the poor performance of SPO 0.3’s worst run



(a) CMAES-Griewangk, 1,000 test runs of CMA-ES used for evaluation

Fig. 15.9: Same as Fig. 8(d), but using 1,000 instead of 100 runs of CMA-ES to evaluate each parameter setting. Note the very poor performance of some CMA-ES runs with the parameter setting from the worst run of SPO 0.4. For this parameter setting, the cloud of points with much higher solution cost than any of the other runs contains 14 points

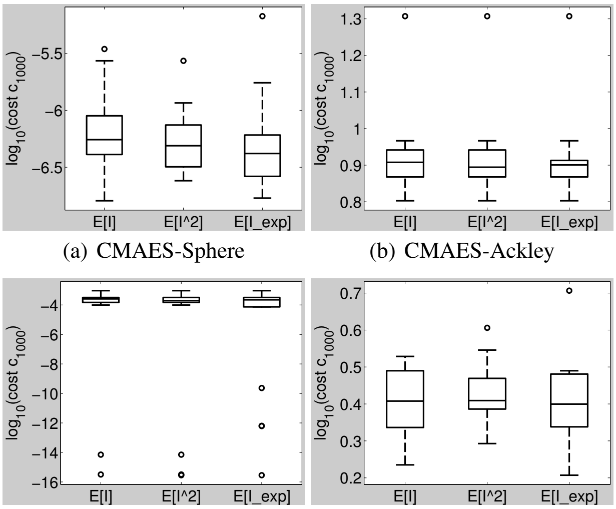
set performance. Recall that this test set performance is the mean solution quality across 100 test runs of CMA-ES. To obtain an independent estimate of a parameter setting's true performance, we performed an additional 100 test runs of CMA-ES for each of these nine parameter settings, with a set of random number seeds disjoint from those used in the original test set.

In Fig. 15.8, we plot the performance of these nine parameter settings for the 100 new seeds. (The figure also shows a parameter setting IA, found by our interactive approach. We defer its discussion until Sect. 15.6.) For test case CMA-ES-Sphere, all selected parameter settings performed very similarly. In test case CMA-ES-Ackley, SPO$^+$ seemed to perform slightly better than SPO 0.3 and 0.4, especially in their respective worst runs.

In test case CMA-ES-Rastrigin, note that most selected parameter settings yielded comparable performance, with the exception of the one identified in the worst repetition of SPO 0.3. With several runs whose performance was about five orders of magnitude worse than the remaining runs, this parameter setting was responsible for the poor mean performance across the 25 runs indicated in Fig. 7(d). Note that the SPO 0.3 run returning this poor parameter setting selected it just before running out of its budget of 1,000 calls to CMA-ES. Likewise, one mostly poor SPO 0.4 run selected a good setting just before reaching the limit on target algorithm runs. As we can see in Fig. 7(d), SPO$^+$ performed much more robustly.

In test case CMA-ES-Griewangk, the situation is somewhat more complicated. While Fig. 7(c) shows very sensitive behavior of SPO 0.3 and SPO 0.4, we do not see any evidence for this in Fig. 8(d). Since the evaluation in Fig. 8(d) used 100 different random seeds (different from both the initial random seeds used during parameter optimization and from the "test" random seeds used to produce Fig. 8(d)) there was no guarantee of obtaining similar performance. Indeed, the measured performances for the parameter settings from the worst run of SPO 0.3 and 0.4 were quite different than previously observed; while they yielded poor mean performance before, they did quite well based on the new set of 100 test seeds. We repeated the evaluation with a larger set of 1,000 random seeds and show the result in Fig. 15.9. In this experiment, 14 of the 1,000 CMA-ES runs for the worst repetition of SPO 0.4 showed extremely poor performance. For the setting from the worst run of SPO 0.3, even these 1,000 test runs did not explain the poor performance in Fig. 7(c). To study this further, we performed $100,000$ additional test runs for this setting and found that nine of them yielded similarly poor performance as the 14/1000 poor CMA-ES for the setting above (best function values around 10^2). Another roughly 8500 runs yielded results around 10^{-2} and the rest (about $91,500$ runs) yielded results $< 10^{-10}$. The optimization of target algorithms with such multimodal result distributions requires a large number of target algorithm runs: in the case above, a very sensitive setting performed just as well as a robust one based on as many as 1,000 runs of the target algorithm.

15.5.2 Expected Improvement Criterion



(a) CMAES-Sphere (b) CMAES-Ackley
(c) CMAES-Griewangk (d) CMAES-Rastrigin

Fig. 15.10: Comparison of different expected improvement criteria for optimizing the performance of CMA-ES on our standard benchmarks. We show boxplots of performance c_{1000} achieved in the 25 runs of each optimizer for each test case

In sequential model-based optimization, parameter settings to be investigated are selected based on an expected improvement criterion (EIC). This aims to address the exploration/exploitation trade-off between learning about new, unknown parts

Table 15.5: The p-values for pairwise comparisons of different expected improvement criteria for optimizing the performance of CMA-ES on our standard benchmarks. These p-values correspond to the data in Fig. 15.10 and are based on a pairwise Max-Wilcoxon test as described in Sect. 15.2

	Sphere	Ackley	Griewangk	Rastrigin
$E[I]$ *vs* $E[I^2]$	0.29	0.55	**0.016**	0.90
$E[I]$ *vs* $E[I_{\text{exp}}]$	0.63	0.25	0.11	**0.030**
$E[I^2]$ *vs* $E[I_{\text{exp}}]$	0.54	0.32	0.77	0.38

of the parameter space and intensifying the search locally in the best known region. We briefly summarize two common versions of the EIC, and then describe a novel variation that we also investigated.

The classic expected improvement criterion used by Jones et al. (1998) is defined as follows. Given a deterministic function f and the minimal value f_{min} seen so far, the improvement at a new parameter setting $\boldsymbol{\theta}$ is defined as

$$I(\boldsymbol{\theta}) := \max\{0, f_{min} - f(\boldsymbol{\theta})\}. \tag{15.9}$$

Of course, this quantity cannot be computed, since $f(\boldsymbol{\theta})$ is unknown. We therefore compute the expected improvement, $E[I(\boldsymbol{\theta})]$. To do so, we require a probabilistic model of f, in our case the GP model. Let $\mu_{\boldsymbol{\theta}} := E[f(\boldsymbol{\theta})]$ be the mean and $\sigma^2_{\boldsymbol{\theta}}$ be the variance predicted by our model, and define $u := (f_{min} - \mu_{\boldsymbol{\theta}})/\sigma_{\boldsymbol{\theta}}$. Then one can show that $E[I(\boldsymbol{\theta})]$ has the following closed-form expression:

$$E[I(\boldsymbol{\theta})] = \sigma_{\boldsymbol{\theta}} \times [u \times \Phi(u) + \varphi(u)], \tag{15.10}$$

where φ and Φ denote the probability density function and cumulative distribution function of a standard normal distribution, respectively.

A generalized expected improvement criterion was introduced by Schonlau et al. (1998), who considered the quantity

$$I^g(\boldsymbol{\theta}) := \max\{0, [f_{min} - f(\boldsymbol{\theta})]^g\} \tag{15.11}$$

for $g \in \{0, 1, 2, 3, \ldots\}$, with larger g encouraging more global search behavior. The value $g = 1$ corresponds to the classic EIC. SPO uses $g = 2$, which takes into account the uncertainty in our estimate of $I(\boldsymbol{\theta})$, since $E[I^2(\boldsymbol{\theta})] = (E[I(\boldsymbol{\theta})])^2 + \text{Var}(I(\boldsymbol{\theta}))$ and can be computed by the closed-form formula

$$E[I^2(\boldsymbol{\theta})] = \sigma^2_{\boldsymbol{\theta}} \times [(u^2 + 1) \times \Phi(u) + u \times \varphi(u)]. \tag{15.12}$$

One issue that seems to have been overlooked in previous work is the interaction between log-transformations of the data and the EIC. When we use a log transformation, we do so in order to increase predictive accuracy, yet our loss function continues to be defined in terms of the untransformed data (e.g., actual runtimes). Hence we should optimize the criterion

$$I_{\exp}(\boldsymbol{\theta}) := \max\{0, f_{min} - e^{h(\boldsymbol{\theta})}\}, \tag{15.13}$$

where $h(\cdot)$ predicts log performance and f_{min} is the untransformed best known function value.

Let $v := (\ln(f_{min}) - \mu_{\boldsymbol{\theta}})/\sigma_{\boldsymbol{\theta}}$. Then, we have the following closed-form expression (see the appendix for the proof):

$$E[I_{\exp}(\boldsymbol{\theta})] = f_{min}\Phi(v) - e^{\frac{1}{2}\sigma^2_{\boldsymbol{\theta}}+\mu_{\boldsymbol{\theta}}} \times \Phi(v - \sigma_{\boldsymbol{\theta}}). \tag{15.14}$$

In Fig. 15.10 and Table 15.5, we experimentally compare SPO$^+$ with these three expected improvement criteria on the CMA-ES test cases, based on a random LHD and log-transformed data. Overall, the differences are small. On average, $E[I^2]$ yielded the best results for test case CMA-ES-sphere, and our new criterion $E[I_{\exp}]$ performed best in the remaining cases. Even though not visually obvious from the boxplots, 2 of the 12 pairwise differences were statistically significant based on a Max-Wilcoxon test.

15.5.3 Overall Evaluation

Table 15.6: Comparison of final performance of various parameter optimization procedures for optimizing SAPS on instance QWH. We report mean ± standard deviation of performance c_{20000} (median search steps SAPS required on instance QWH in 1,000 test runs using the parameter settings the method chose after $20,000$ algorithm runs), across 25 repetitions of each method. Based on a Mann-Whitney U test, SPO$^+$ performed significantly better than CALIBRA, BasicILS, FocusedILS, and SPO 0.3 with p-values 0.015, 0.0002, 0.0009, and 4×10^{-9}, respectively. The p-value for a comparison against SPO 0.4 was 0.06

Procedure	SAPS median runtime [search steps]
SAPS default from Hutter et al. (2002)	85.5×10^3
CALIBRA(100) from Hutter et al. (2007)	$10.7 \times 10^3 \pm 1.1 \times 10^3$
BasicILS(100) from Hutter et al. (2007)	$10.9 \times 10^3 \pm 0.6 \times 10^3$
FocusedILS from Hutter et al. (2007)	$10.6 \times 10^3 \pm 0.5 \times 10^3$
SPO 0.3	$18.3 \times 10^3 \pm 13.7 \times 10^3$
SPO 0.4	$10.4 \times 10^3 \pm 0.7 \times 10^3$
SPO$^+$	$\mathbf{10.0 \times 10^3 \pm 0.4 \times 10^3}$

In Sects. 15.5.1 and 15.5.2, we fixed the design choices of using log transformations and initial designs based on random LHDs. Now, we revisit these choices. Using our new SPO$^+$ intensification criterion and expected improvement criterion $E[I^2]$, we studied how much the final performance of SPO$^+$ changed when not using a log-transformation and when using different methods to create the initial design. Not surprisingly, none of the initial designs led to significantly better final performance than any of the others. The result for the log transformation was

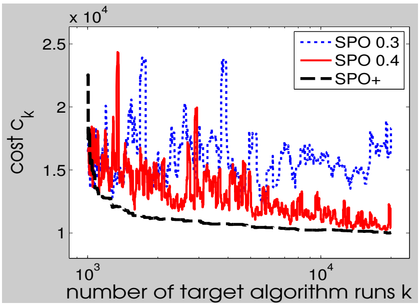


Fig. 15.11: Comparison of SKO and two variants of SPO (discussed in Sect. 15.5.1) for optimizing CMA-ES on the Sphere function. Comparison of SPO variants (all based on a random LHD and log-transformed data) for minimizing SAPS median runtime on instance QWH. We plot the solution cost c_k of each method (median search steps SAPS required on instance QWH in 1,000 test runs using the parameter settings the method chose after k algorithm runs), as a function of the number of algorithm runs, k, it was allowed to perform. These values are averaged across 25 runs of each method

more surprising. Although we saw in Sect. 15.4 that the log transformation consistently improved predictive model performance, based on a Mann-Whitney U test it turned out to significantly improve *final* parameter optimization performance only for CMA-ES-sphere.

Finally, we compared the performance of SPO 0.3, 0.4, and SPO$^+$ (all based on random LHDs and using log-transformed data) to the parameter optimization methods studied by Hutter et al. (2007). We summarize the results in Table 15.6. While SPO 0.3 performed worse than the other methods, SPO 0.4 performed comparably, and SPO$^+$ outperformed all methods with the exception of SPO 0.4 significantly. Figure 15.11 illustrates the difference between SPO 0.3, SPO 0.4, and SPO$^+$ for this SAPS benchmark. Similar to what we observed for CMA-ES (Fig. 15.7), SPO 0.3 and 0.4 changed their incumbents very frequently, with SPO 0.4 showing more robust behavior than SPO 0.3, and SPO$^+$ in turn much more robust behavior than SPO 0.4.

15.6 Interactive Exploration of Parameter Space

The automated methods discussed so far in this chapter can be very effective when it is possible to perform a relatively large number of evaluations of the given target algorithm to be optimized. However, there are cases in which target algorithm evaluations are overwhelmingly costly compared to the overall computational resources and time available for the parameter optimization process. Real-world applications of this nature can be found in the optimization of engineering designs in the aerospace and automotive industry (Alexandrov et al. 2001), in hydrological applications (Mayer et al. 2002), and in climate modeling. In those examples, the algorithms to be optimized control or model the behavior of complex systems, and evaluating their performance for a single parameter configuration can take hundreds of CPU hours. In such cases, the number of parameter configurations that can be evaluated is severely limited, and approaches that achieve good results based on a small number of evaluations are required. In the following, we explore an interactive approach for model-based parameter optimization that utilizes human judgment and insight in conjunction with statistical methods.

15.6.1 Using SPOT Interactively

The sequential parameter optimization toolbox (SPOT) was developed to improve the performance of algorithms and to gain insight into their working mechanisms (Beielstein 2003). When using SPOT interactively, the experimenter makes use of the statistical analysis techniques supported by the toolbox to sequentially select new settings for the given target algorithm. The general flow of the interactive sequential parameter optimization process is illustrated in Fig. 14.2 on page 346. We will illustrate this interactive approach by means of a case study, in which we focus primarily on the performance of CMA-ES on the Rastrigin function. This function was chosen because the previously studied, fully-automated SPO procedures SPO 0.3 and SPO 0.4 performed relatively poorly compared to SPO$^+$(see, e.g., Fig. 15.1). The interactive approach might shed some light on this poor performance, since it provides tools for understanding the structure of the search space (ROI) and the determination of factor effects. Later, we also report results for CMA-ES on the other three classical test problems from global optimization introduced in Sect. 15.2 (Sphere, Griewangk, and Ackley). We note that, while in principle, the interactive approach can be applied to the optimization of any objective function, the linear regression models used in the following fit arithmetic mean performance. Further, although experimenters working in the context of real-world performance optimization tasks often rely upon prior knowledge about the target algorithm, here we assume, for the sake of generality, that no prior knowledge is available regarding important parameters, optimal experimental designs or regression models (predictors) for the target algorithm. We do, however, assume some general knowledge about reasonable ranges for parameters of evolutionary strategies, such as CMA-ES.

15.6.1.1 Pre-experimental Planning

In the pre-experimental planning phase, we have to select an initial design, a predictor, and a quality measure. Our goal is to improve CMA-ES, which requires the specification of the four parameters NPARENTS, NU, CS, and DAMPS. As a rule of thumb, we assert that the experimenter should not invest more than about a quarter of the available budget (here the number of CMA-ES runs) in the first design.[8] In most circumstances, it is unwise to plan too comprehensive a design at this outset (Box et al. 1978).

In light of our goal of keeping the number of configurations to be evaluated relatively small, we have chosen a *Box–Behnken design* as the initial design (Box and Behnken 1960, Pukelsheim 1993). Box–Behnken designs are central composite designs that augment 2^k designs with center points (see also the discussion of designs in Chap. 3 of this book). Box–Behnken designs can be used to calibrate full quadratic models; furthermore, they are rotatable and, when the number of factors is four or fewer, require fewer runs than central composite designs. (Chapts. 2 and 3 in this book discuss further design considerations.) Box–Behnken designs avoid the corners of the region of interest and allow experimenters to work around extreme factor combinations. This is especially important for the optimization of CMA-ES, because large NPARENTS and large NU values at the same time are expected to produce poor results. The four-factor Box–Behnken design used in the first step of our analysis requires 27 runs (Pukelsheim 1993).

We now discuss how the region of interest (ROI) was determined for our experiments. We note that this determination of the ROI settings is an integral part of the interactive approach and more elaborate than in the previously discussed automated approach. Imagine that a single run of CMA-ES has a budget of $t = 10,000$ function evaluations. (Note that here we discuss the number of function evaluations performed by *every CMA-ES run*, not the number of CMA-ES parameter settings that can be evaluated by the experimenter.) As a rule of thumb, the smallest number of generations for evolutionary algorithms that use some step-length adaptation is $g = 10$. This gives an upper limit for the population size of $NPARENTS \times NU = 1{,}000$, so $NPARENTS_{\max} = 100$ and $NU_{\max} = 10$ are reasonable values. The lower bounds of the region of interest for these two variables was chosen as $NPARENTS_{\min} = 2$ and $NU_{\min} = 2$. Since no information about reasonable settings for CS and DAMPS is known, we have chosen the maximum interval length, i.e., $CS \in [0.1; 1]$ and $DAMPS \in [0.25; 0.99]$. In general, we recommend liberally-chosen intervals in the absence of prior knowledge justifying tighter bounds.

15.6.1.2 Prediction Model

First, we consider a linear model without interactions. We apply a logarithmic transformation because it improves the fit. The transformation is motivated by model di-

[8] Note, however, that there are exceptions to this rule. For example, Bartz-Beielstein and Preuss (2006) state:

> The experimental analysis clearly demonstrated that the determination of a suitable initial design is of crucial importance for the second phase, which performs a local tuning. To play safe, we recommend increasing the number of initial design points. The number of sequential optimization steps could be reduced in many situations without a significant performance loss.

agnostics, e.g., residuals plotted versus predicted values and histogram plots of the data. Our regression modeling is based on coded variables, i.e., $\{-1, 0, +1\}$, rather than on the natural variables of the target algorithm. For example, consider a region of interest for some variable in the range from one to ten. The coded variables $\{-1, 0, +1\}$ correspond to the natural variables $\{1, 5.5, 10\}$. (Transformations and standardizations are discussed in Chaps. 2 and 3 of this book. See also Kleijnen (2008, p.30-31).) The linear regression model is given by

$$y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \epsilon,$$

where the x_i's are explanatory variables (or predictors) representing NPARENTS, NU, CS, and DAMPS, and the β_i's can be estimated using the method of least squares. Further details are presented in the Appendix to this book. Here, y denotes the function value produced by the run of the target algorithm (CMA-ES). The fitted least squares equation we obtained based on data from the first 27 runs was

$$\hat{y} = 3.8 + 0.93x_1 + 1.37x_2 - 1.84x_3 + 0.39x_4.$$

In addition, we also performed visual inspections, e.g., on the basis of added-variable plots (see also Chap. 14). Box and Draper (1987) mention some elementary checks for interaction and curvature. For example, a comparison of the average y_c at the center of the design with the average of the remaining points of the Box–Behnken design y_{-c} gives a measure of the overall curvature of the response surface.

15.6.1.3 Model Selection

Next, we check whether interaction terms should be included into the model. This is done by increasing model complexity in a stepwise manner. R's `stepAIC()` function is used for performing model searches by the *Akaike information criterion* (AIC) (Venables and Ripley 2002). Here, smaller AIC values are better. The function `stepAIC()` can be used for an automated stepwise selection procedure. It requires a fitted model to define the starting process and a list of two formulae defining the most complex and the simplest models. We have chosen the linear model with two-factor interactions as the most complex model, and the model which includes the four main factors only as the simplest model. The automated search leads to a model which, in addition to the four main factors, includes interactions between NPARENTS and NU, DAMPS and CS, and NU and CS. Figure 15.12 shows the output from this analysis. We note that it is easy to be misled by this automated model search, and experience shows that different variables could be selected if the `stepAIC()` procedure were repeated on a new, similar data set (Dalgaard 2002). In many cases, the decision between models cannot be based on the data alone, but should take into consideration results from previous investigations or theoretical considerations. Therefore, it is recommended that users carefully evaluate results obtained by the `stepAIC()` procedure.

```
Stepwise Model Path
Analysis of Deviance Table
Initial Model:
Y ~ NPARENTS + NU + DAMPS + CS
Final Model:
Y ~ NPARENTS + NU + DAMPS + CS + NPARENTS:NU + DAMPS:CS +
    NU:CS
           Step Df Deviance Resid. Df Resid. Dev      AIC
1                                 22   57.95982 30.62566
2 + NPARENTS:NU  1 7.082627        21   50.87720 29.10660
3  + CS:DAMPS  1 5.881372        20   44.99582 27.78979
4     + NU:CS  1 4.350343        19   40.64548 27.04437
```

Fig. 15.12: Output from R's `stepAIC()` procedure

In the second step of the model selection process, we analyze the enhanced model from the point of view of regression-based significance. Here, we apply R's `dropterm()` function to the final model from the `stepAIC()` procedure. Venables and Ripley (2002) note that selecting the terms on the basis of the `stepAIC()` criterion can be somewhat permissive in its choice of terms. We also observed this in our own analysis (Fig. 15.13).

```
Call: lm(formula = Y ~ NPARENTS + NU + DAMPS + CS + NPARENTS:NU,
    data = df0012normlogy)
Residuals:
     Min       1Q   Median       3Q      Max
-3.79993 -0.79931  0.02917  1.04553  2.12640
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   3.8001     0.2996  12.686 2.59e-11 ***
NPARENTS      0.9279     0.4493   2.065 0.051483 .
NU            1.3686     0.4493   3.046 0.006142 **
DAMPS        -1.8354     0.4493  -4.085 0.000531 ***
CS          0.3886     0.4493   0.865 0.396912
NPARENTS:NU   1.3307     0.7783   1.710 0.102040
---
Residual standard error: 1.557 on 21 degrees of freedom
Multiple R-squared: 0.6175,     Adjusted R-squared: 0.5264
F-statistic:  6.78 on 5 and 21 DF,  $p$-value: 0.0006601
```

Fig. 15.13: Result from R's `dropterm()` analysis. Starting point for this analysis is the model proposed by the stepwise selection method `stepAIC()`

We observe that the regression coefficients for NPARENTS, NU, and DAMPS are large relative to their standard errors. Based on the conventional significance level from the regression analysis, i.e., $Pr(> |t|)$, we conclude that the model `Y` $\sim$ `NPARENTS + NU + DAMPS` should be used for the steepest descent. Predictions of this model are illustrated in Fig. 15.14. The contour lines can be tentatively accepted as a rough estimate of the underlying response function over the region of interest explored so far. (Further details of the model selection based on the t-statistics are discussed on p. 431 of the Appendix to this book.)

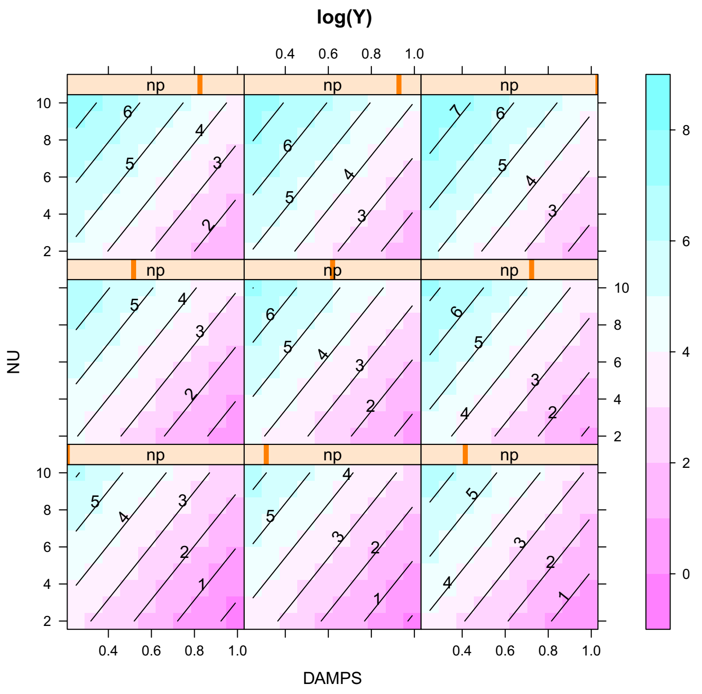


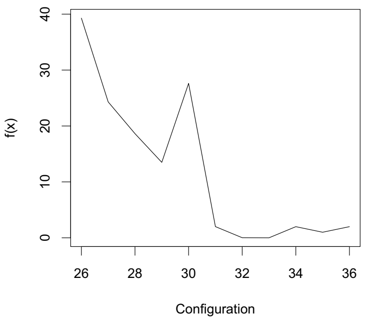
Fig. 15.14: Data from the first design (27 runs). These data were used to determine the steepest descent. These contour plots support the assumption that the values for DAMPS should be increased, and values for NU should be decreased, whereas the impact of the population size (NPARENTS) is relatively small. Predicted values are based on the regression equation $\hat{y} = 3.8 + 0.93x_1 + 1.37x_2 - 1.84x_3$. Values for NPARENTS are taken from the interval $[2, 100]$, which was split into nine subintervals. The slider on top of each panel indicates the value of population size

15.6.1.4 Steepest Descent

We proceed with the steepest descent based on the model `Y ~ NPARENTS + NU + DAMPS + CS` (note that we included factor `CS`; although this factor was not judged to have a significant impact by the regression model, including it also did not require any additional effort). The procedure of steepest descent is performed as described on p. 354 in Chap. 14. Again, as the largest step width we recommend the value that leads to the border of the ROI. Here, we obtained a step width of $\delta x_4 = 0.036$ in the natural variables for DAMPS, leading to 11 design points until we hit the border of the ROI (DAMPS reaches its maximal value 0.99). Data from the steepest descent experiments are shown in Table 15.7. A graph of these results to determine the new region of interest following the direction of the steepest descent is shown in Fig. 15.15.

Table 15.7: Steepest descent experiment

	Y	NPARENTS	NU	TCCS	DAMPS	CONFIG
1	39.29	51	6.00	0.55	0.62	26
2	24.31	49	5.70	0.54	0.66	27
3	18.63	46	5.40	0.53	0.69	28
4	13.49	44	5.11	0.52	0.73	29
5	27.61	41	4.81	0.51	0.77	30
6	2.00	39	4.51	0.50	0.81	31
7	0.01	36	4.21	0.49	0.84	32
8	0.00	34	3.91	0.48	0.88	33
9	1.99	31	3.61	0.47	0.92	34
10	0.99	29	3.32	0.46	0.95	35
11	1.99	26	3.02	0.45	0.99	36



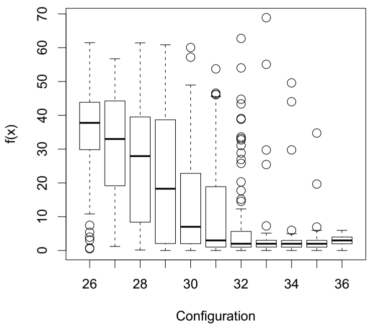


Fig. 15.15: *Left:* Function values $f(x)$ (Rastrigin) versus steps along the path of the steepest descent. Indices denote the eleven steps from the steepest descent. Note that these values are based on one repeat only, so variation in the data, e.g., the peak (index 5), is not surprising. These data were used during the interactive approach. *Right:* Boxplots showing the variance of the data used for the steepest descent. Same situation as on the *left*, but 100 repeats of each CMA-ES configuration. These experiments were performed after the CMA-ES tuning was finished. Information from these runs was not used to determine the tuned CMA-ES parameter set

These settings are used for additional runs of CMA-ES. Figure 15.15 plots the yield at each step along the path of the steepest descent. Based on visual inspection of the yields in Fig. 15.15, the new central point was determined to be the eighth point of the steepest descent since no significant decrease occurred during steps nine through eleven. Furthermore, this new central point leaves some space for variation of the DAMPS values, say, in the interval $[0.8, 0.99]$.

15.6.1.5 Second Model and Steepest Descent

Based on the best value obtained with the steepest descent, we build a new model with center point

$$\mathbf{x}_c = [NPARENTS, NU, CS, DAMPS] = [34, 3.91, 0.48, 0.88].$$

The specification of the new region of interest requires user knowledge. The new center point was determined by interpreting graphical results based on the steepest descent. Next, we have to determine a new region around $\mathbf{x}_c$. Sometimes, especially when a classical factorial design is used during the first step, it can be useful to increase the region of interest at this stage. However, we have chosen a Box–Behnken design for the first step and therefore have to decrease the region of interest. As a rule of thumb, to be reconsidered on a case-by-case basis, we use at least $\pm$ $1/5$th of the values at the new central point. For example, if the value of the new population size NPARENTS is 50, we define a new region of interest for this values as the interval $[40, 60]$. Here, the new region of interest reads as follows: $NPARENTS \in [24, 44]$, $NU \in [3, 5]$, $CS \in [0.4, 0.6]$, and $DAMPS \in [0.8, 0.99]$.

Again, a Box–Behnken design with 27 points is used to set up a regression model to determine the path of the steepest descent. Steps along this path are performed until no improvement is obtained. Note that five repeats are used now (previous experiments used only one repeat). The path among the steepest descents consists of 14 steps. Thus, 70 experiments are performed. The final configuration reads

$$[NPARENTS, NU, CS, DAMPS] = [34, 3, 0.43, 0.98].$$

15.6.1.6 Final Exploration

Finally, we use graphical tools to get an overview of the experimental region used in our experiments. Figure 15.16 displays a fit of the response surface which is based on the complete data set. A local regression model based on R's `loess()` function is fitted to the data.

Altogether 135 (= 27 + 11 + 27 + 70) runs of CMA-ES were used. The experiments were performed on a 2.3 GHz Pentium 4 with 4 GB RAM running MATLAB Version 7.6 on a Linux system. The SPOT runs for the interactive exploration

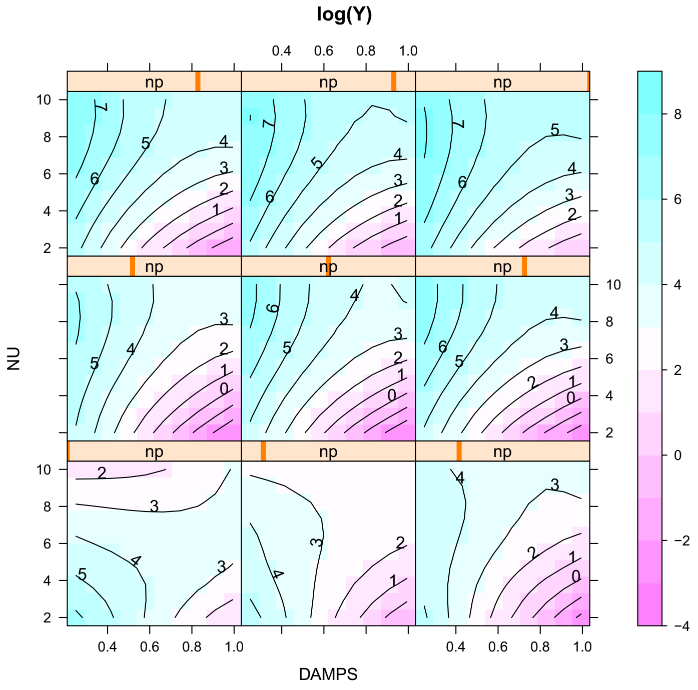


Fig. 15.16: Contour plot based on the complete data set (CMAES-Rastrigin with 135 function evaluations). Smaller values are better. Better configurations are placed in the lower right corner of the panels. The factor which has the smallest effect, CS, is held constant. NU is plotted versus DAMPS, while values of the factor with the second smallest effect, namely NPARENTS (np), are varied with the slider on top of each panel

required 39 seconds. Writing the reports and setting up the R scripts for the interactive exploration took approximately one hour. Our experience from working on real-world problems indicates that one working day is necessary to perform the complete SPOT process if applied to a new simulation or optimization algorithm. This includes discussions with domain experts to define performance criteria and the specification and implementation of SPOT interfaces. Substantially more time might be required in cases where target algorithm runs are very costly.

15.6.2 Further Interactive Tuning Results

We also applied our interactive tuning approach to the other three CMA-ES scenarios introduced in Sect. 15.2. Here we briefly summarize the results.

15.6.2.1 Interactive Tuning of CMA-ES on the Sphere Function

Our experiments with CMA-ES on the Sphere function used a different setup (i.e., different values of starting point, dimension, and number of function evaluations); see Table 15.2. However, regarding the region of interest, the same initial settings as reported in Sect. 15.6.1.1 were used. Again, a Box–Behnken design was generated, and 27 runs of CMA-ES were performed. Based on the results from these runs, the following regression model was fitted:

$$\hat{y} = 2.13 + 9.25x_1 + 1.13x_2 + 0.003x_3 - 0.74x_4.$$

The regression analysis revealed that the value for NPARENTS should be decreased. The regression model as described in Sect. 15.6.1.2 and its refinement by steepest descent produced

$$[NPARENTS, NU, CS, DAMPS] = [2.0, 3.5, 0.6, 0.9].$$

Note the drastic change in the NPARENTS values, whereas other predictors are only slightly modified. Here, following the direction of the steepest descent resulted in a significant improvement of CMA-ES's performance. The function value could be reduced from 7132 to 1.40×10^{-5}, and—in a repeat of the steepest descent—from 5862 to 1.27×10^{-6}. Since we reached a region with small function values and relatively little variation, we decided to stop the procedure at this stage and perform a visual inspection based on contour plots (see Fig. 15.17). This required no additional function evaluations.

The final configuration for the sphere function reads

$$[NPARENTS, NU, CS, DAMPS] = [2.0, 3.5, 0.6, 0.9].$$

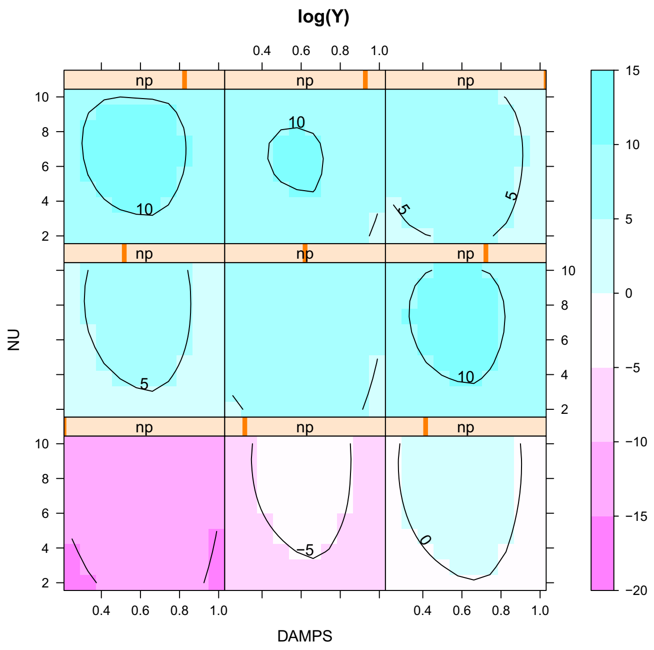
Altogether 49 (= 27 + 2 $\times$ 11) runs of CMA-ES were used to produce this result.

15.6.2.2 Interactive Tuning of CMA-ES on the Ackley Function

The same initial settings as reported in Sect. 15.6.1.1 were used for the interactive tuning of the Ackley function. Again, we generated a Box–Behnken design and performed 27 runs of CMA-ES. Based on the results from these runs, the following regression model was obtained (through fitting, as previously described):

$$\hat{y} = 1.75 + 0.88x_2 + 0.21x_3 - 0.61x_4.$$

Compared to the fit of the functions considered so far, where the regression model showed p-values smaller than 0.01 (e.g., a value of 0.0008 in the first regression model), the fit of the regression model was relatively poor (p-value 0.2). This behavior can be explained as follows. Running CMA-ES on Ackley's function with parameters from the initial ROI produces many outliers which disturb the model-



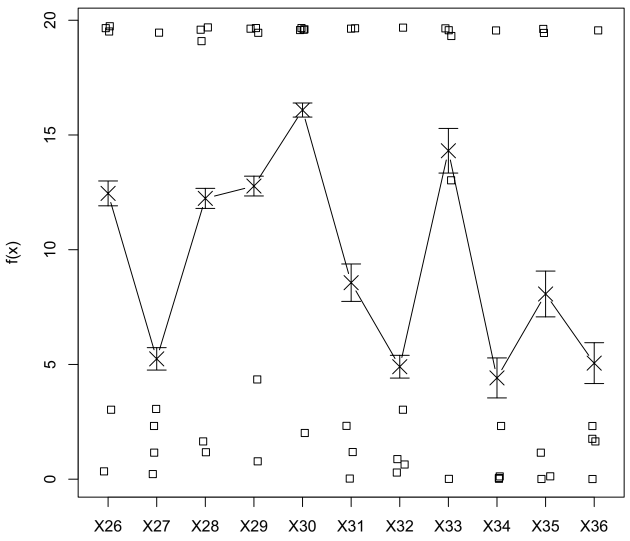


Fig. 15.17: *Top:* Contour plot based on the complete data set (CMAES-Sphere, 49 data points). Smaller values are better. The factor which has the smallest effect, CS, is held constant. NU is plotted versus DAMPS, while values of the factor with the last but one effect, namely NPARENTS (np), are varied with the slider on top of each panel. *Bottom:* Function values $f(x)$ (Ackley) versus steps along the path of the steepest descent. Indices denote the eleven steps from the steepest descent. Note that these values are based on five repeats

ing process. CMA-ES generates good (< 5) and bad (> 15) solutions with the same parameter setting. The SPOT tuning procedure applied in this study is based on mean values. Figure 15.17 illustrates difficulties arising from this situation. The RSM is based on mean values and produces unhelpful gradient information. There is a relatively large gap between good and bad solutions. Note that the mean lies exactly in this gap, so it represents a value that is never realized. We believe that in this case tuning mean performance might not yield the most meaningful results. Instead, one could use the trimmed distribution; this will be subject of forthcoming studies.

We nevertheless present results from the interactive approach to complete our study. A contour plot of the predicted function values of the CMA-ES for parameter values from the region of interest used in the interactive tuning study of the Ackley function is shown in Fig. 15.18. The final configuration reads

$$[NPARENTS, NU, CS, DAMPS] = [2, 2, 0.11, 0.55].$$

As in the case of the Sphere function, 27 initial runs were performed, followed by the evaluations on the path of the steepest descent (2$\times$ 11). To obtain more insight, we performed three additional runs during the steepest descent (3 $\times$ 11). Therefore, a total of 82 runs (= 27 + 2$\times$11 + 3 $\times$ 11) was used in this experiment.

15.6.2.3 Interactive Tuning of CMA-ES on the Griewangk Function

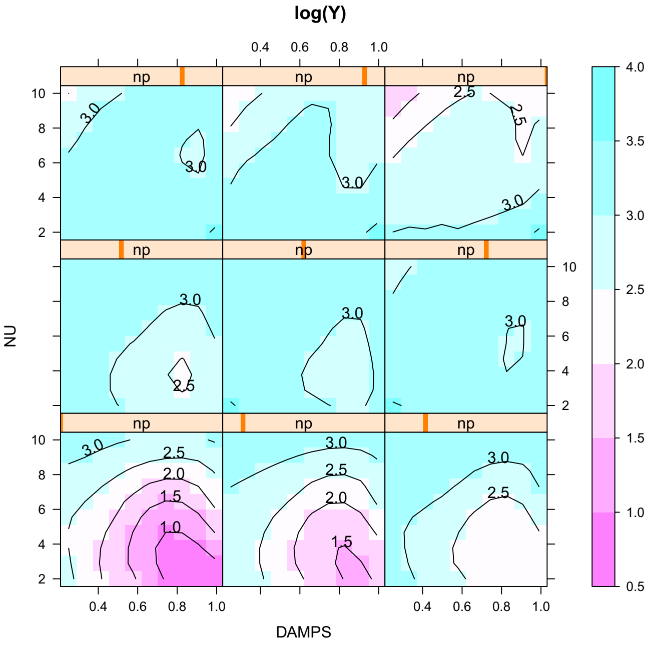
The same initial settings as reported in Sect. 15.6.1.1 was used. Again, a Box–Behnken design was generated, and 27 runs of CMA-ES were performed. Based on the results from these runs, the following regression model was fitted:

$$\hat{y} = -5.65 + 0.61x_1 + 9.07x_2 - 1.12x_3 - 4.30x_4.$$

A steepest descent (with two repeats) based on this first regression model led directly to an improved CMA-ES configuration. To validate the improvement following the path of the steepest descent, we repeated the corresponding runs twice. The improved configuration reads

$$[NPARENTS, NU, CS, DAMPS] = [48, 2, 0.61, 0.80].$$

Since $49 = 27 + (11 \times 2)$ runs of CMA-ES already resulted in an improved configuration, we used 20 additional CMA-ES runs to obtain an overview of the region of interest, scanning this region based on Latin hypercube sampling. The corresponding contour plot is shown in Fig. 15.18. As for the Sphere function, 27 initial runs were performed, followed by the evaluations on the path of the steepest descent (2$\times$ 11). Therefore, a total of 69 runs (27 + 2$\times$11 + 20) was used in this experiment.



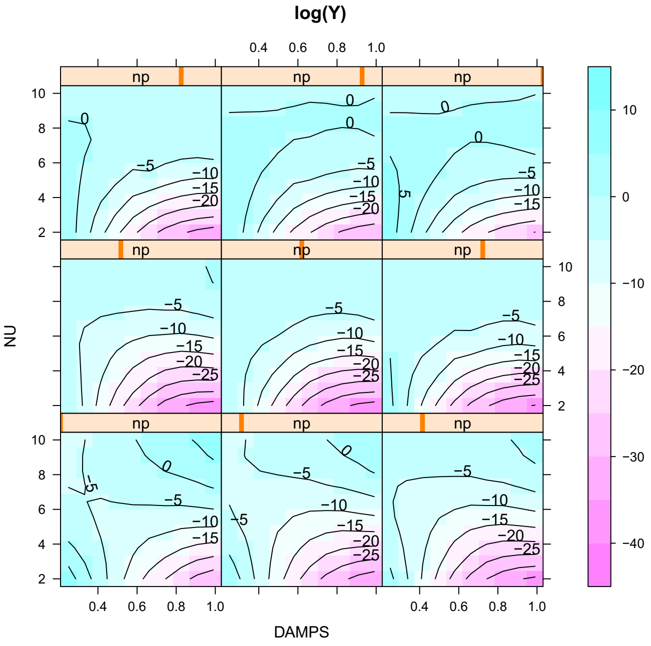


Fig. 15.18: *Top:* Contour plots based on the complete data set (CMA-ES Ackley based on 82 function evaluations). Smaller values are better. Better configurations are placed in the lower area of the panels. The factor CS is held constant. NU is plotted versus DAMPS, while values of the factor NPARENTS (np), are varied with the slider on top of each panel. *Bottom:* Contour plots based on the complete data set (CMA-ES Griewangk based on 69 function evaluations). Smaller values are better. Better configurations are placed in the lower region, i.e., small NU values are important. The factor which has the smallest effect, CS, is held constant. NU is plotted versus DAMPS, while values of the factor with the last but one effect, namely NPARENTS (np), are varied with the slider on top of each panel

Table 15.8: Performance comparison of the parameter settings found by our automatic and interactive approaches. We give median and mean of solution costs c_{1000} across the 25 repetitions of SPO$^+$. For each function, we also give the number of target algorithm runs, K, used in the interactive approach and the resulting solution cost, c_K, as well as the quantile corresponding to this value in the empirical distribution of SPO$^+$ results (e.g., 56% means that 14 of 25 SPO$^+$ runs yielded better results)

Test	**25 repetitions of SPO$^+$**		**Interactive approach**		
case	median	mean ± stddev	K	sol. cost	quantile of SPO$^+$ dist.
Sphere [$\times 10^{-7}$]	4.16	5.55 ± 5.15	49	4.65	56%
Ackley	7.97	8.48 ± 2.61	82	12.25	96%
Griewangk [$\times 10^{-4}$]	1.73	2.66 ± 2.53	69	2.22	56%
Rastrigin	2.50	2.62 ± 0.51	135	2.82	68%

15.6.3 Comparison of Solutions Found Automatically and Interactively

Next, we evaluate how the interactively found parameter settings compare to the automatically found ones in terms of CMA-ES performance achieved. Table 15.8 compares the performance of the manually identified parameter settings against the distribution of performance achieved across 25 runs of SPO$^+$. In all test cases, the median-best SPO$^+$ run achieved better performance than the manual procedure; between 56% (14/25) and 96% (24/25) of the automatically found parameter settings performed better than the one identified manually. However, recall that the manual process used many function evaluations less than the automatic procedure. Nevertheless, on two test cases, the manually identified parameter settings performed better than the automatically found settings do on average. We also provide boxplots for the performance of the interactively found settings in Fig. 15.8; their performance is comparable to the one of the automatically found settings.

We conclude that often a manually executed classical regression analysis can yield well-performing parameter settings using a very limited number of runs of the target algorithm. The exception in our experiments was test case CMA-ES-Ackley. For this test case, we observed a pronounced multimodal distribution during the interactive tuning; we hypothesize that this caused problems for the classical regression analysis. Although the distributions for test cases CMA-ES-Rastrigin and CMA-ES-Griewangk were also multimodal, the poor runs were much rarer in these test cases and rather played the role of outliers. (In the boxplots of Fig. 15.8 on page 389, the poor runs in test cases CMA-ES-Rastrigin and CMA-ES-Griewangk were indeed marked as outliers, while the multimodal distributions for CMA-ES-Ackley were not seen as caused by outliers).

15.6.4 Discussion of the Interactive Approach

Similar to microscopes in biology, SPOT can be used as a "datascope" to gain insight into algorithm behavior, by revealing factor effects and their importance to the experimenter. Such insights can not only be used to guide the interactive parameter optimization process, but also be of intrinsic value to the developer or end user of a target algorithm.

The classical response surface methodology (as discussed in Chap. 15 of Box et al. (1978)) underlying our interactive approach was developed not only for finding parameter settings that achieve improved performance, but also to provide insights into how the performance of a target algorithm is affected by parameter changes. This latter question is related to the analysis of the response surface in the region of interest, and contour plots as shown in Fig. 15.16 are useful tools to answer it.

In particular, from the results reported earlier in this section, we can conclude that CMA-ES performs robustly on the test functions we studied in the sense that its mean performance (e.g., as summarized by contour plots) varies only moderately with changes in the parameters. We furthermore observed that large DAMPS values and smaller NU values resulted in better CMA-ES performance, while the effect of CS was rather marginal. Finally, small population sizes improved CMA-ES's performance on the Sphere function, corresponding nicely with theoretical results for evolution strategies (Schwefel 1995, Beyer 2001). These statements can be understood as hypotheses derived from our experimental results, and each of them could be further studied by additional experiments, e.g., as described on p. 34 in Chap. 2 of this book.

In our case study illustrating the interactive approach, we used classical regression models, because these models can be interpreted quite easily; features of the response surface can be seen directly from the regression equation $Y = X\beta$. This is not the case for more sophisticated prediction models, such as neural networks or Gaussian process models. Furthermore, as demonstrated here in the case of CMA-ES, it is possible to obtain competitive results using such simple models. Nevertheless, in principle, more complex regression models could be used in the context of the interactive sequential parameter optimization approach. Furthermore, we note that observations and hypotheses regarding the dependence of a given target algorithm's performance on its parameter settings could also be obtained by analyzing more complex models, including the Gaussian process models constructed by the previously discussed, automatic sequential parameter optimization procedures.

Clearly, the interactive approach makes it easy to use results from early stages of the sequential parameter optimization process to effectively guide decisions made at later stages. For example, looking back at the initial stages of the process, the experimenter can detect that the set of variables studied at this stage was chosen poorly, or that inappropriate ranges were chosen for certain variables. Box et al. (1978) state: "It is rather like looking at an old movie of a swimmer, who can now do back flips from a high diving board, when he was a young child making his first feeble attempts to keep his head above water. [...] The investigator must learn from the swimmer, who was prepared to begin by putting his foot in the water and was not afraid of getting wet." We note that the models used in early stages of the automated procedures discussed earlier in this chapter also provide guidance to later stages of the process. However, the interactive process leaves room for expert human judgment, which can often be more effective in terms of the improvement achieved based on a small number of target algorithm runs.

The human expertise required to use the interactive approach successfully can be seen as a drawback compared to fully automated approaches. However, by providing dedicated support for the various operations that need to be carried out in this context, SPOT eases the burden on the experimenter and lowers the barrier to using the interactive approach effectively.

15.7 Conclusions and Future Work

In this work, we experimentally investigated model-based approaches for optimizing the performance of parametrized, randomized algorithms. First, we restricted our attention to procedures based on GP models, the most popular family of models for this problem. We evaluated two approaches from the literature, and found that "out-of-the-box" sequential parameter optimization (SPO) offered more robust performance than the sequential Kriging optimization (SKO) approach. However, when a log-transformation was used, SKO performed competitively. We then investigated key design decisions within the SPO paradigm: the initial design; whether to fit models to raw or log-transformed data; the expected improvement criterion; and the intensification criterion. Of these four, the log transformation and the intensification criterion substantially affected performance. Based on our findings, we proposed a new version of SPO, dubbed SPO$^+$, which yielded substantially better performance than SPO for optimizing the solution quality of CMA-ES (Hansen and Ostermeier 1996, Hansen and Kern 2004) on a number of test functions, as well as the runtime of SAPS (Hutter et al. 2002) on a SAT instance. In this latter domain, for which performance results for other (model-free) parameter optimization approaches are available, we demonstrated that SPO$^+$ achieved state-of-the-art performance.

We then contrasted this automated tuning approach with an interactive approach based on classical linear regression models. The interactive approach yielded well-performing parameter settings based on very few function evaluations, and also provided the basis for interesting hypotheses about CMA-ES's performance under different parameter settings. The interactive approach is particularly suitable in situations where the evaluation of individual configurations is computationally very expensive and therefore the overall number of parameter configurations evaluated has to be kept as low as possible.

In the future, we plan to extend our work to deal with optimization of runtime across a set of instances, along the lines of the approach of Williams et al. (2000). We also plan to compare other types of models, such as random forests (Breiman 2001), to the Gaussian process approach. SPOT and the interactive approach already support the optimization of categorical parameters using tree-based regression models (Chap. 14). We further plan to develop automated methods for the sequential optimization of categorical variables.

Acknowledgements We thank Theodore Allen for making the original SKO code available to us. This work was supported by NSERC Discovery Grant 238788, by the Bundesministerium für Forschung und Bildung (BMBF) under the grant FIWA (AIF FKZ 17N2309, "Ingenieurnachwuchs"), and by the Cologne University of Applied Sciences under the grant COSA.

Appendix

We show that for a random variable X distributed according to a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, it is the case that $E[\max(f_{min} - \exp(X), 0)] = f_{min}\Phi(v) - e^{\frac{1}{2}\sigma^2+\mu} \times \Phi(v-\sigma)$, where $v = \frac{\ln(f_{min})-\mu}{\sigma}$. We denote the probability density function and cumulative distribution function of a standard normal distribution as φ and Φ, respectively.

$$
\begin{aligned}
& E[\max(f_{min} - \exp(X), 0)] \\
&= \int_{-\infty}^{\infty} \max(f_{min} - \exp(x), 0) p(x) dx \\
&= \int_{-\infty}^{\ln(f_{min})} (f_{min} - \exp(x)) \frac{1}{\sigma}\varphi(\frac{x-\mu}{\sigma}) dx \\
&= f_{min}\Phi(\frac{\ln(f_{min})-\mu}{\sigma}) - \int_{-\infty}^{\frac{\ln(f_{min})-\mu}{\sigma}} \exp[x\sigma + \mu] \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{1}{2}x^2\right] dx \\
&= f_{min}\Phi(\frac{\ln(f_{min})-\mu}{\sigma}) - \int_{-\infty}^{\frac{\ln(f_{min})-\mu}{\sigma}} \exp[\frac{1}{2}\sigma^2 + \mu] \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{1}{2}(x-\sigma)^2\right] dx \\
&= f_{min}\Phi(\frac{\ln(f_{min})-\mu}{\sigma}) - \exp[\frac{1}{2}\sigma^2 + \mu]\Phi(\frac{\ln(f_{min})-\mu}{\sigma} - \sigma).
\end{aligned}
$$

References

Adenso-Diaz B, Laguna M (2006) Fine-tuning of algorithms using fractional experimental design and local search. Operations Research 54(1):99–114

Alexandrov NM, Lewis RM, Gumbert CR, Green LL, Newman PA (2001) Approximation and model management in aerodynamic optimization with variable-fidelity models. Journal of Aircraft 38(6):1093–1101

Audet C, Orban D (2006) Finding optimal algorithmic parameters using the mesh adaptive direct search algorithm. SIAM Journal on Optimization 17(3):642–664

Balaprakash P, Birattari M, Stützle T (2007) Improvement strategies for the f-race algorithm: Sampling design and iterative refinement. In: Bartz-Beielstein T, Aguilera MJB, Blum C, Naujoks B, Roli A, Rudolph G, Sampels M (eds) 4th International Workshop on Hybrid Metaheuristics (HM'07), pp 108–122

Bartz-Beielstein T (2003) Experimental analysis of evolution strategies—overview and comprehensive introduction. Interner Bericht des Sonderforschungsbereichs 531 Computational Intelligence CI–157/03, Universität Dortmund, Germany

Bartz-Beielstein T (2006) Experimental Research in Evolutionary Computation—The New Experimentalism. Natural Computing Series, Springer, Berlin, Heidelberg, New York

Bartz-Beielstein T, Markon S (2004) Tuning search algorithms for real-world applications: A regression tree based approach. In: Greenwood GW (ed) Proceedings 2004 Congress on Evolutionary Computation (CEC'04), Portland OR, IEEE, Piscataway NJ, vol 1, pp 1111–1118

Bartz-Beielstein T, Preuss M (2006) Considerations of budget allocation for sequential parameter optimization (SPO). In: Paquete L, et al. (eds) Workshop on Empirical Methods for the Analysis of Algorithms, Proceedings, Reykjavik, Iceland, pp 35–40

Bartz-Beielstein T, Parsopoulos KE, Vrahatis MN (2004a) Analysis of particle swarm optimization using computational statistics. In: Simos TE, Tsitouras C (eds) Proceedings International Conference Numerical Analysis and Applied Mathematics (ICNAAM), Wiley-VCH, Weinheim, Germany, pp 34–37

Bartz-Beielstein T, Parsopoulos KE, Vrahatis MN (2004b) Design and analysis of optimization algorithms using computational statistics. Applied Numerical Analysis and Computational Mathematics (ANACM) 1(2):413–433

Bartz-Beielstein T, de Vegt M, Parsopoulos KE, Vrahatis MN (2004c) Designing particle swarm optimization with regression trees. Interner Bericht des Sonderforschungsbereichs 531 Computational Intelligence CI–173/04, Universität Dortmund, Germany

Bartz-Beielstein T, Lasarczyk C, Preuß M (2005) Sequential parameter optimization. In: McKay B, et al. (eds) Proceedings 2005 Congress on Evolutionary Computation (CEC'05), Edinburgh, Scotland, IEEE Press, Piscataway NJ, vol 1, pp 773–780

Bartz-Beielstein T, Lasarczyk C, Preuss M (2008a) Sequential parameter optimization toolbox, manual version 0.5, September 2008, available at `http://www.gm.fh-koeln.de/imperia/md/content/personen/lehrende/bartz_beielstein_thomas/spotdoc.pdf`

Bartz-Beielstein T, Zimmer T, Konen W (2008b) Parameterselektion für komplexe modellierungsaufgaben der wasserwirtschaft – moderne CI-verfahren zur zeitreihenanalyse. In: Mikut R, Reischl M (eds) Proc. 18th Workshop Computational Intelligence, Universitätsverlag, Karlsruhe, pp 136–150

Beachkofski B, Grandhi R (2002) Improved distributed hypercube sampling. American Institute of Aeronautics and Astronautics Paper 2002-1274

Beielstein T (2003) Tuning evolutionary algorithms—overview and comprehensive introduction. Interner Bericht des Sonderforschungsbereichs 531 *Computational Intelligence* CI–148/03, Universität Dortmund, Germany

Beyer HG (2001) The Theory of Evolution Strategies. Springer, Berlin, Heidelberg, New York

Birattari M, Stützle T, Paquete L, Varrentrapp K (2002) A racing algorithm for configuring metaheuristics. In: Proc. of GECCO-02, pp 11–18

Box G, Behnken D (1960) Some new three level designs for the study of quantitative variables. Technometrics 2:455–475

Box GEP, Draper NR (1987) Empirical Model Building and Response Surfaces. Wiley, New York NY

Box GEP, Hunter WG, Hunter JS (1978) Statistics for Experimenters. Wiley, New York NY

Breiman L (2001) Random forests. Machine Learning 45(1):5–32

Chen J, Chen C, Kelton D (2003) Optimal computing budget allocation of indifference-zone-selection procedures, working paper, taken from `http://www.cba.uc.edu/faculty/keltonwd`. Cited 6 January 2005

Coy SP, Golden BL, Runger GC, Wasil EA (2001) Using experimental design to find effective parameter settings for heuristics. Journal of Heuristics 7(1):77–97

Dalgaard P (2002) Introductory Statistics with R. Springer, Berlin, Heidelberg, New York

Hansen N (2006) The CMA evolution strategy: a comparing review. In: Lozano J, Larranaga P, Inza I, Bengoetxea E (eds) Towards a new evolutionary computation. Advances on estimation of distribution algorithms, Springer, pp 75–102

Hansen N, Kern S (2004) Evaluating the CMA evolution strategy on multimodal test functions. In: Yao X, et al. (eds) Parallel Problem Solving from Nature PPSN VIII, Springer, LNCS, vol 3242, pp 282–291

Hansen N, Ostermeier A (1996) Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In: Proc. of CEC-96, Morgan Kaufmann, pp 312–317

Hoos HH, Stützle T (2005) Stochastic Local Search – Foundations & Applications. Morgan Kaufmann

Huang D, Allen TT, Notz WI, Zeng N (2006) Global optimization of stochastic black-box systems via sequential kriging meta-models. Journal of Global Optimization 34(3):441–466

Hutter F, Tompkins DAD, Hoos HH (2002) Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In: Proc. of CP-02, pp 233–248

Hutter F, Hamadi Y, Hoos HH, Leyton-Brown K (2006) Performance prediction and automated tuning of randomized and parametric algorithms. In: Proc. of CP-06, pp 213–228

Hutter F, Hoos HH, Stützle T (2007) Automatic algorithm configuration based on local search. In: Proc. of AAAI-07, pp 1152–1157

Hutter F, Hoos HH, Leyton-Brown K, Murphy KP (2009a) An experimental investigation of model-based parameter optimisation: SPO and beyond. In: Proc. of GECCO-09, pp 271–278

Hutter F, Hoos HH, Leyton-Brown K, Stützle T (2009b) ParamILS: an automatic algorithm configuration framework. Journal of Artificial Intelligence Research 36:267–306

Ihaka R, Gentleman R (1996) R: A language for data analysis and graphics. Journal of Computational and Graphical Statistics 5(3):299–314

Jones DR, Schonlau M, Welch WJ (1998) Efficient global optimization of expensive black box functions. Journal of Global Optimization 13:455–492

Kleijnen JPC (2008) Design and analysis of simulation experiments. Springer, New York NY

Konen W, Zimmer T, Bartz-Beielstein T (2009) Optimierte Modellierung von Füllständen in Regenüberlaufbecken mittels CI-basierter Parameterselektion. at – Automatisierungstechnik 57(3):155–166

Lasarczyk CWG (2007) Genetische Programmierung einer algorithmischen Chemie. PhD thesis, Technische Universität Dortmund

Leyton-Brown K (2003) Resource allocation in competitive multiagent systems. PhD thesis, Stanford University

Leyton-Brown K, Nudelman E, Shoham Y (2002) Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In: Proc. of CP-02

Lophaven SN, Nielsen HB, Sondergaard J (2002) Aspects of the Matlab toolbox DACE. Tech. Rep. IMM-REP-2002-13, Informatics and Mathematical Modelling, Technical University of Denmark, DK-2800 Kongens Lyngby, Denmark

Mayer AS, Kelley C, Miller CT (2002) Optimal design for problems involving flow and transport phenomena in saturated subsurface systems. Advances in Water Resources 12:1233–1256

Mockus J, Tiesis V, Zilinskas A (1978) The application of bayesian methods for seeking the extremum. Towards Global Optimisation 2:117–129, north Holland, Amsterdam

Montgomery DC (2001) Design and Analysis of Experiments, 5th edn. Wiley, New York NY

Pukelsheim F (1993) Optimal Design of Experiments. Wiley, New York NY

Quinonero-Candela J, Rasmussen CE, Williams CK (2007) Approximation methods for gaussian process regression. In: Large-Scale Kernel Machines, Neural Information Processing, MIT Press, Cambridge, MA, USA, pp 203–223, URL `http://mitpress.mit.edu/9780262026253`

Rasmussen CE, Williams CKI (2006) Gaussian Processes for Machine Learning. The MIT Press

Sacks J, Welch WJ, Welch TJ, Wynn HP (1989) Design and analysis of computer experiments. Statistical Science 4(4):409–423

Santner TJ, Williams BJ, Notz WI (2003) The Design and Analysis of Computer Experiments. Springer Verlag, New York

Schonlau M, Welch WJ, Jones DR (1998) Global versus local search in constrained optimization of computer models. In: Flournoy N, Rosenberger W, Wong W (eds) New Developments and Applications in Experimental Design, vol 34, Institute of Mathematical Statistics, Hayward, California, pp 11–25

Schwefel HP (1995) Evolution and Optimum Seeking. Sixth-Generation Computer Technology, Wiley, New York NY

Tompkins DAD, Hoos HH (2004) UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT & MAX-SAT. In: Proc. of SAT-04

Venables WN, Ripley BD (2002) Modern Applied Statistics with S-PLUS, 4th edn. Springer, Berlin, Heidelberg, New York

Williams BJ, Santner TJ, Notz WI (2000) Sequential design of computer experiments to minimize integrated response functions. Statistica Sinica 10:1133–1152

Appendix

Appendix A
A Brief Introduction to Inferential Statistics

Dario Basso

Abstract This appendix introduces the elements of statistical theory that are used throughout the book. It starts by defining random variables and the theoretical models to describe them. It then briefly outlines the concepts underlying point estimation. The central part is dedicated to hypothesis testing and confidence intervals by means of which inference from sample statistics to population parameters is carried out. Subsequently, the treatment focuses on regression and modeling, which play a fundamental role in several chapters of this book. The presentation is necessarily limited to linear regression and to basic model fitting. For a broader treatment of statistical theory the reader is referred to any textbook of statistics and to Mood et al. (1974) and Davison (2008).

A.1 Introduction

Inferential statistics is a collection of techniques that allow us to deduce information from a set of observed data about a phenomenon under investigation in a *population* of interest.

The motivation for such techniques is the fact that surveying the phenomenon on the entire population (census) in order to gain complete knowledge of it is often unrealistic or too expensive. An alternative is then to extract a set of *statistical units* (i.e., a the phenomenon on this restricted set of units. We may do this because we expect the units in the sample to reproduce the phenomenon as it is in the population (with a certain degree of *variability* depending on the *size* of the sample).

A "well-chosen" sample should be representative of the population, therefore the statistical units to be included in the sample should be chosen independently of the

Dario Basso
Department of Statistics, University of Padua, Italy
e-mail: dario@stat.unipd.it

characteristics of the phenomenon in the population. This is usually known as a *random sampling* from a population.

There are several ways of sampling units from a population (e.g., with replacement or without replacement), and we may consider finite populations (e.g., the non-isomorphic graphs of a certain size for which we wish to determine the chromatic number) or nonfinite populations (such as the runtime of an optimization algorithm).

In inferential statistics, the observed data are usually considered as n independent realizations of a random experiment, whose possible outcomes are described by a *random variable* (r.v.).[1] The distribution of the random variable depends on some unknown parameters of the population and on how the sample has been extracted. We can also say that the random variable is a *model* of the phenomenon in the population, and that each datum is a realization of the same random variable. According to this model, the sample data are then considered as n realizations of *independent* and *identically distributed* (i.i.d.) random variables.

In *parametric* inference, the random variable describing the experiment is defined by a *probabilistic model*, which is usually a mathematical formula determining the probability of an event (or a set of events). The probabilistic model is identified by a parameter θ (or sometimes by a vector of parameters) that takes values in a *parameter space* Θ. The collection of probabilistic models identified by all possible values of θ in Θ is called a *parametric statistical model*. Here the inference on the phenomenon in the population is translated into an inference on the unknown parameter θ that identifies a specific distribution among those that belong to the statistical model. In the *likelihood* function approach, we seek the value for θ that is most in agreement (*likely*) with the observed data. There are other approaches to inference, such as nonparametric and Bayesian inference.

This probabilistic approach serves two purposes: (i) describing the variability of the sample outcomes and (ii) evaluating the uncertainty of the inference, e.g., by providing an interval of possible values for the true parameter θ, or by evaluating the risk of incorrectly answering the question "does the true parameter θ belong to a certain subset $\Theta_0 \in \Theta$?".

Let us end this paragraph with an explanatory example: suppose that we have a deterministic program, for example, a mixed integer programming solver, and that we want to determine its ability to solve a specific class of instances of a certain optimization problem, say the set covering problem.[2] Let θ be the true, unknown proportion of instances that can be solved within a runtime of, say $t_0 = 1{,}000$ s, that is, the amount of time we are prepared to accept before giving up. Suppose that a random sample of n instances is taken from the class of instances, and that the application of the solver to each instance of the sample is coded into two possible

[1] A random variable is a function assigning a real number to each element of a probability space.

[2] In the optimization version of the set covering problem, we are given a universe $\mathcal{U}$ and a family $\mathcal{S}$ of subsets of $\mathcal{U}$, and we want to find a cover, that is, a subfamily $\mathcal{C} \subseteq \mathcal{S}$ of sets whose union is $\mathcal{U}$, that minimizes the number of selected sets. A class of instances can be determined, for example, by specifying a range for the number of objects in $\mathcal{U}$ and for the number of subsets in $\mathcal{S}$ and a certain structure in the elements covered by the subsets. Then, the class of instances, although large, is a finite set.

outcomes: 0 meaning "not solved within t_0", and 1 meaning "solved." We can thus describe each outcome with a random variable X_i assuming the values 0 and 1 with probability θ and $1-\theta$, $i = 1,\dots,n$. We know from probability calculus that the random variable S, the sum of n independent and identically distributed dichotomous variables (such as X_i), has a binomial distribution whose probability function is

$$\Pr\{S=s\} = p_S(s;\theta) = \binom{n}{s}\theta^s(1-\theta)^{(n-s)} \qquad s \in \{0,1,\dots,n\}, \quad \theta \in (0,1).$$

Then the binomial distribution is the probabilistic model describing the outcome (sum) of the sample data, whereas the statistical model is the set $\mathcal{P} = \{p_S(s;\theta), \theta \in (0,1)\}$. The initial goal evaluating the proportion of solvable instances, is then translated into estimating the unknown parameter θ.

Once an estimate of θ has been obtained, it will be possible, through a specified probabilistic model, to answer questions such as: "is $\theta > 90\%$?" "Given that, if we change sample (i.e., if we repeat the experiment) we will have a different result, can we say something about the uncertainty of θ (i.e., can we give a set of reasonable values for θ)?" Of course, each of the previous questions cannot be answered with certainty. Inferential statistics can answer the previous questions while determining the probability of "incorrect" conclusions on θ.

A.1.1 Random Variables

A univariate quantitative random variable (r.v.) X is a variable taking values in a domain D_X with prespecified probabilities. There are two kinds of quantitative r.v.s: *discrete* and *continuous* (or absolutely continuous). In the former case the cardinality of the support is at most numerable (i.e., it has almost the same cardinality of $\mathbb{N}$), in the latter $D_X \subseteq \mathbb{R}$.

An r.v. X is therefore defined by the specification of the domain D_X and the probability associated with each possible outcome of X. Two very intuitive examples are the following. The outcome of a fair die is characterized by an r.v. X with $D_X = \{1,2,3,4,5,6\}$ and the probability associated with each outcome (also called the *realization* of X) is 1/6. The launch of a fair coin can be described by the r.v. X assuming two values (or *modalities*): "head" and "tail", each with probability 1/2. This last example actually refers to a *categorical* variable (i.e., an r.v. whose possible outcomes are categories or adjectives), that can be recoded in order to obtain a discrete one (e.g., taking values in $D_X = \{0,1\}$, with 0 being "head" and 1 being "tail").

As far as discrete r.v.s are concerned, the probability of the event $\{X = x\}$, $x \in D_X$, is given by the *probability function* $p_X(x)$ is summarized by a mathematical formula. For instance, if X is dichotomous and the probability of the event $\{X = 1\}$ is denoted by the parameter $\theta \in (0,1)$, then $p_X(x) = \theta^x(1-\theta)^{(1-x)}$. This distribution is known as the *Bernoulli* distribution. The probability function satisfies

$0 \leq p_X(x) \leq 1$ for all $x \in D_X$, where 0 denotes the probability of an impossible event, and 1 that of an (almost) sure event.

Another important function that is related to the r.v.s is the *cumulative distribution function* (cdf), or *distribution function*, which is defined as

$$F_X(x) = \Pr\{X \leq x\}, \qquad x \in \mathbb{R}.$$

Note that $F_X(x)$ is a continuous, nonnegative, nondecreasing function that satisfies $\lim_{x \to -\infty} F_X(x) = 0$ and $\lim_{x \to +\infty} F_X(x) = 1$. The probability of the event $X \in [a, b]$ can be computed as $F_X(b) - F_X(a)$.

In the continuous case, the event $\{X = x\}$ has zero probability for all $x \in D_X$, so the probability function does not apply here. The r.v. is then described by the *density function* $f_X(x)$, which is defined as

$$f_X(x) = \lim_{\delta \to 0} \frac{\Pr\{X \leq x + \delta\} - \Pr\{X \leq x\}}{\delta} = \frac{\partial F_X(x)}{\partial x}, \quad \delta > 0.$$

Therefore the relationship between $F_X(x)$ and $f_X(x)$ can be defined as

$$F_X(x) = \int_{-\infty}^{x} f_X(t)\mathrm{d}t \qquad x \in \mathbb{R}.$$

Of course, a similar definition can be applied to the discrete case by letting

$$F_X(x) = \sum_{t \leq x} \Pr\{X = t\}, \qquad x \in \mathbb{R}.$$

Note that, when X is discrete, its cdf is typically a stepfunction.

The functions $p_X(x)$ and $f_X(x)$ are nonnegative and must, respectively, satisfy

$$\sum_{x \in D_X} p_X(x) = 1, \qquad \int_{x \in D_X} f_X(x)\mathrm{d}x = 1.$$

Usually, the cdf can also be specified by a closed mathematical formula.

The distribution of an r.v. is characterized by some indexes. One of them is the *expected value*, which is defined as

$$\mathrm{E}[X] = \sum_{x \in D_X} x p_X(x) \qquad \text{if } X \text{ is discrete}$$

$$\mathrm{E}[X] = \int_{x \in D_X} x f_X(x)\mathrm{d}x \quad \text{if } X \text{ is continuous.}$$

The expected value (some aliases are *expectation*, *mean of the distribution*, *first moment*) is a linear operator, i.e., $\mathrm{E}[a + bX] = a + b\mathrm{E}[X]$. For instance, the expected value of a fair die is equal to 3.5; the expected value of a Bernoulli variable X with $P\{X = 1\} = \theta$ is equal to $\mathrm{E}[X] = 1 \cdot \theta + 0 \cdot (1 - \theta) = \theta$. Note that $\mathrm{E}[X]$ is *not* an r.v., and it can sometimes be one of the parameters of the distribution.

Another possible parameter of a probability distribution is the *variance*, which is defined as

$$\begin{aligned}\mathrm{V}[X] &= \mathrm{E}[X - \mathrm{E}[X]]^2 = \mathrm{E}[X^2 - 2X\mathrm{E}[X] + \mathrm{E}[X]^2] \\ &= \mathrm{E}[X^2] - 2\mathrm{E}[X]^2 + \mathrm{E}[X]^2 = \mathrm{E}[X^2] - \mathrm{E}[X]^2.\end{aligned}$$

Thus, the variance of X, when X is the outcome of a fair die is equal to $91/6 - 3.5^2 = 2.917$. If X is a Bernoulli variable with $P\{X = 1\} = \theta$, then $\mathrm{E}[X^2] = 1^2 \cdot \theta + 0^2 \cdot (1 - \theta) = \theta$, and therefore $\mathrm{V}[X] = \theta - \theta^2 = \theta \cdot (1 - \theta)$. Note that $\mathrm{V}[X] > 0$ (since $\mathrm{V}[X] = 0$ implies that X is actually a constant). The variance is also known as the *second central moment*. The expected value and variance may be nonfinite. For instance, the Cauchy distribution, whose (standard) density function is $f_X(x) = (1+x^2)^{-1}$, does not admit finite moments. There are other distributions that do not admit finite moments for some values of their parameters. One of them is the *Pareto distribution*, whose density function is

$$f_X(x; x_0, \theta) = \frac{\theta x_0^\theta}{x^{\theta+1}} \qquad x \geq x_0 > 0;\ \theta > 0.$$

For the Pareto distribution $\mathrm{E}[X] = \theta x_0/(1 - \theta)$, which exists when $\theta > 1$, and $\mathrm{E}[X^2] = x_0^2\theta/(2 - \theta)$, which exists when $\theta > 2$. In general, if $\mathrm{E}[X^r]$ is finite, then all the moments of order s with $s < r$ are also finite.

Other useful indicators of an r.v. are the *quantiles*. A quantile of order α, $\alpha \in (0, 1)$ is the *modality* x_α of an r.v. whose cdf satisfies $F_X(x_\alpha) = \alpha$. Note that, if X is continuous, there is a one-to-one relationship between $x_\alpha \in D_X$ and $\alpha \in (0, 1)$ (see uniform distribution). A very special quantile is the *median* of a distribution, defined as the quantile of order 1/2. Therefore, the *median* is the modality $x_{0.5}$ that satisfies $F_X(x_{0.5}) = 1/2$. Note the quantiles are always well defined only if X is continuous.

A.1.2 Examples of Statistical Models

In this section we review a few statistical models that are used in this book. The first two models are for discrete random variables while all the others are for continuous random variables. The description is necessarily concise; for extensive treatment see Johnson and Kotz (1970).

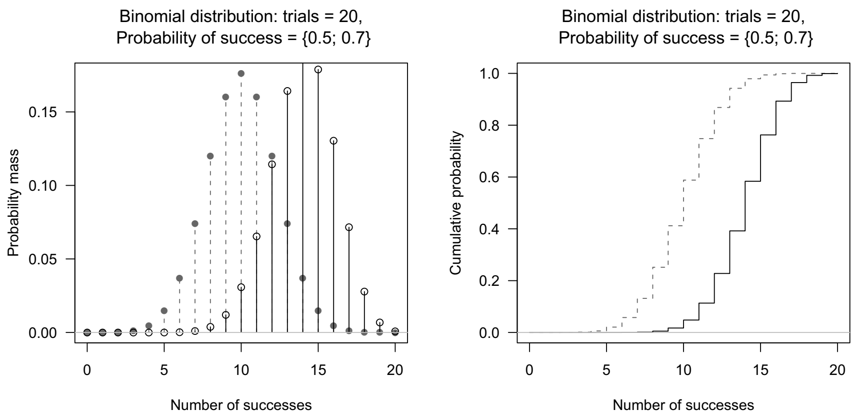


Fig. A.1: The binomial distribution for $n = 20$ and $\theta = 0.5$ (dashed line) or $\theta = 0.7$ (full line)

Binomial Random Variable

In the previous pages we encountered already the Bernoulli r.v.s. A *Binomial* r.v. is the sum of n i.i.d. *Bernoulli* r.v.s. We may indicate a random variable X with Bernoulli distribution using the notation $X \sim \mathrm{Bi}(1, \theta)$. Then, the notation for the Binomial is $X \sim \mathrm{Bi}(n, \theta)$. Its probability and distribution functions are, respectively,

$$p_X(x) = \binom{n}{x}\theta^x(1-\theta)^{n-x}, \qquad F_X(x) = \Pr\{X \le x\} = \sum_{i=0}^{x}\binom{n}{i}\theta^i(1-\theta)^{n-i},$$

and are shown in Fig. A.1. The mean of the binomial distribution is $\mathrm{E}[X] = n\theta$. The variance of the distribution is $\mathrm{V}[X] = n\theta(1-\theta)$ (see next section).

Poisson Random Variable

The *Poisson* r.v. is used in modeling random arrivals. In this case we can see X as the number of arrivals in one unit of time and hence $D_X = \mathbb{N}$.

The probability function is

$$p_X(x) = \mathrm{e}^{-\lambda}\frac{\lambda^x}{x!}, \qquad \lambda \in \mathbb{R}^+,$$

and is identified by the parameter $\lambda > 0$, which is the mean of X. Figure A.2 shows an example with $\lambda = 1$. We denote this model by $X \sim Po(\lambda)$. To see that $\sum_{x\in D_X} p_X(x) = 1$, obtain Taylor's expansion of the function $\exp\{\lambda x\}$ in $x_0 = 0$, and let $x = 1$. The sum of n independent Poisson r.v.s with parameters $\lambda_1, \lambda_2, \ldots, \lambda_n$ is still a Poisson r.v. with parameter $\sum_{i=1}^n \lambda_i$.

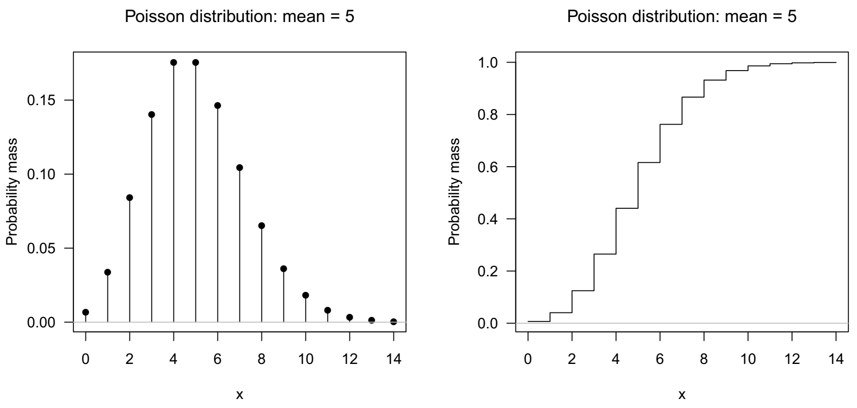


Fig. A.2: The Poisson distribution for $\lambda = 1$

Uniform Random Variable

This variable is defined in the interval $[a, b]$. We write it as $X \sim U[a, b]$. Its density and cumulative distribution functions are, respectively,

$$f_X(x) = \frac{I_{[a,b]}(x)}{b-a}, \qquad F_X(x) = \frac{1}{b-a}\int_{-\infty}^{x} I_{[a,b]}(t)\mathrm{d}t = \frac{x-a}{b-a},$$

where $I_{[a,b]}(\cdot)$ is the indicator function of the interval $[a, b]$. See Fig. A.3. Note that, if we set $a = 0$ and $b = 1$, we obtain $F_X(x) = x$, $x \in [0, 1]$. A typical example is the following: the cdf of a continuous r.v. is uniformly distributed in $[0, 1]$. The proof of this statement is as follows: For $u \in [0, 1]$, we have

$$\begin{aligned}\Pr\{F_X(X) \le u\} &= \Pr\{F_X^{-1}(F_X(X)) \le F_X^{-1}(u)\} = \Pr\{X \le F_X^{-1}(u)\} \\ &= F_X(F_X^{-1}(u)) = u.\end{aligned}$$

This means that, when X is continuous, there is a one-to-one relationship (given by the cdf) between $x \in D_X$ and $u \in [0, 1]$.

Normal (or Gaussian) Random Variable

This variable is defined on the support $D_X = \mathbb{R}$ and its density function is given by

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}.$$

The density function is identified by the pair of parameters (μ, σ^2), where $\mu \in \mathbb{R}$ is the mean (or location parameter) and $\sigma^2 > 0$ is the variance (or dispersion param-

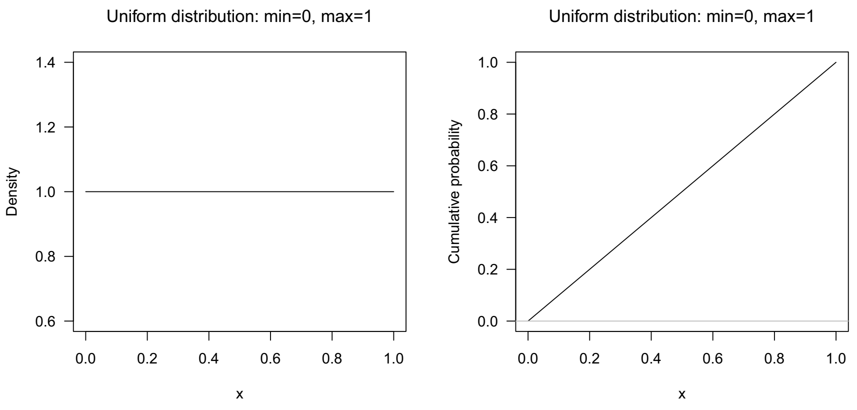


Fig. A.3: The uniform distribution in the interval $[0, 1]$

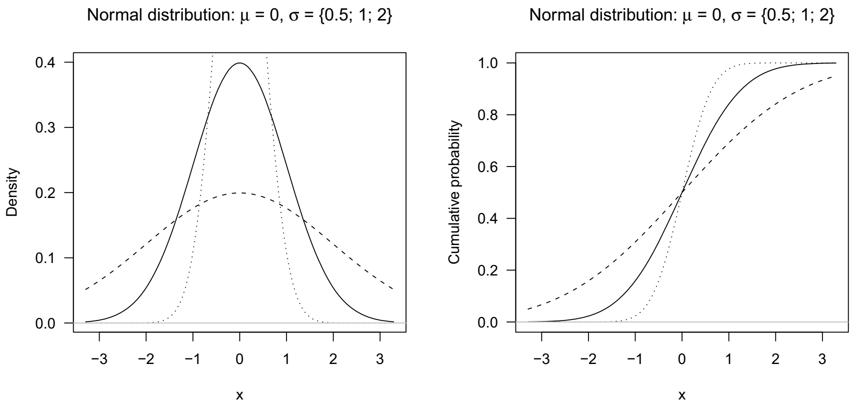


Fig. A.4: The normal distribution for $\sigma^2 = \{0.5; 1; 2\}$ (dotted, full, dashed line, respectively)

eter) of X. The density function is symmetric around μ. Some example of normal densities are given in Figure A.4 for different values of σ^2.

The normal distribution belongs to the location-scale family distributions. This means that, if $Z \sim N(0, 1)$ (read, Z has a standard normal distribution; i.e., with $\mu = 0$ and $\sigma^2 = 1$), and we consider the linear transformation $X = \mu + \sigma Z$, then $X \sim N(\mu, \sigma^2)$ (read, X has a normal distribution with mean μ and variance σ^2). This means that one can obtain the probability of any interval $(-\infty, x]$, $x \in \mathbb{R}$ for any normal distribution (i.e., for any pair of the parameters μ and σ) once the quantiles of the standard normal distribution are known. Indeed

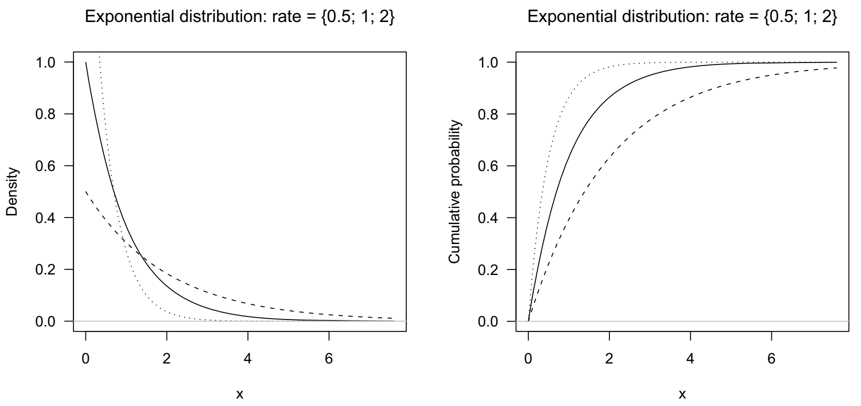


Fig. A.5: The exponential distribution for $\lambda = \{0.5, 1, 2\}$ (dashed, full, dotted line, respectively)

$$\begin{aligned} F_X(x) &= \Pr\{X \leq x\} = \Pr\left\{\frac{X-\mu}{\sigma} \leq \frac{x-\mu}{\sigma}\right\} \\ &= \Pr\left\{Z \leq \frac{x-\mu}{\sigma}\right\} = F_Z\left(\frac{x-\mu}{\sigma}\right) \qquad x \in \mathbb{R}. \end{aligned}$$

The quantiles of the standard normal distribution are available in any statistical program. The density and cumulative distribution function of the standard normal r.v. at point x are usually denoted by the symbols $\phi(x)$ and $\Phi(x)$.

Exponential Random Variable

This is defined on the support $(0, +\infty)$. We write it as $X \sim \mathrm{Exp}(\lambda)$. The density and distribution functions are:

$$f_X(x) = \lambda \mathrm{e}^{-\lambda x}, \qquad F_X(x) = 1 - \mathrm{e}^{-\lambda x} \qquad \lambda > 0$$

and are shown in Fig. A.5. The mean of this distribution is equal to $1/\lambda$; the variance is equal to $1/\lambda^2$. There is a useful reparameterization of this density function which is called *reparameterization with the mean* and can be obtained by letting $\lambda = 1/\theta$; we write this $X \sim \mathrm{Exp}(1/\theta)$. It is easy to prove that the mean and variance of the distribution, according to this reparameterization, are $\mathrm{E}[X] = \theta$ and $\mathrm{V}[X] = \theta^2$. The exponential distribution is used to describe the times at which random arrivals occur. Relevant in this context is the memoryless property, that is, $\Pr\{X > s + x \mid X > s\} = \Pr\{X > x\}$ for all $s, x \geq 0$. Hence, for exponentially distributed arrivals, the probability that we have to wait x seconds for a new arrival, after we had waited s seconds, is not different from the probability that we wait x seconds. The similarity between random arrivals and runtime of stochastic algorithms led to attempts to use this model and its theoretical consequences also in this latter field.

Gamma Random Variable

The exponential distribution is a special case of the *Gamma* distribution. Random variables with this distribution have density function:

$$f_X(x) = \frac{\lambda^\nu x^{\nu-1} \mathrm{e}^{-\lambda x}}{\Gamma(\nu)} \qquad \nu, \lambda > 0, \qquad \Gamma(\nu) = \int_0^{+\infty} \lambda^\nu t^{\nu-1} \mathrm{e}^{-\lambda t} \mathrm{d}t.$$

Here λ is the *scale* parameter and ν the *shape* parameter. We write it as $X \sim \mathrm{Ga}(\nu, \lambda)$. The *Gamma function* $\Gamma(\nu)$ is a standardizing constant, as it satisfies $\lim_{x \to +\infty} F_X(x) = 1$. Moreover, it satisfies the property $\Gamma(\nu+1) = \nu\Gamma(\nu)$, therefore if ν is integer, $\Gamma(\nu) = (\nu - 1)!$.

It can be shown that the mean of the Gamma r.v. is ν/λ and the variance is ν/λ^2. Another important property of the Gamma distribution is that the sum of n independent Gamma r.v.s with the same scale parameter λ and shape parameters ν_i, $i = 1, \ldots, n$ is still distributed as a Gamma r.v. with scale parameter λ and shape parameter $\sum_{i=1}^n \nu_i$

Weibull Random Variable

This is defined on $(0, +\infty)$; its density and distribution functions are

$$f_X(x) = \lambda\nu(\lambda x)^{\nu-1} \exp\{-(\lambda x)^\nu\}, \qquad F_X(x) = 1 - \exp\{-(\lambda x)^\nu\}.$$

and are shown in Fig. A.6. We write it as $X \sim \mathrm{We}(\nu, \lambda)$ and it can be shown that $\mathrm{E}[X] = 1/\lambda\Gamma\,(1 + 1/\nu)$ and $\mathrm{V}[X] = \lambda^{-2}[\Gamma(1 + 2/\nu) - \Gamma(1 + 1/\nu)^2]$.

The exponential distribution can be seen as a special case of the Weibull distribution when $\nu = 1$.

Note that it is always possible to translate the distribution of a r.v. by applying the transformation $Y = X - \mu$. This transformation does not affect the variance and the shape parameters, but affects the support D_Y. By applying this transformation to the examples of Gamma and Weibull r.v.s the support becomes $D_Y = (\mu, +\infty)$.

A.2 Point Estimation

In the previous section, we have seen that the core point of the inferential process is the choice of an adequate statistical model, which is identified by some parameters. In order to model the observed data, the estimation of the parameters of the probability distribution is required. There are several ways to obtain the estimate of the parameters of interest. For instance, moment estimation consists of estimating the parameter(s) of interest with the equivalent sample quantity. Let us assume that the phenomenon under study can be modeled with an r.v. X with unknown param-

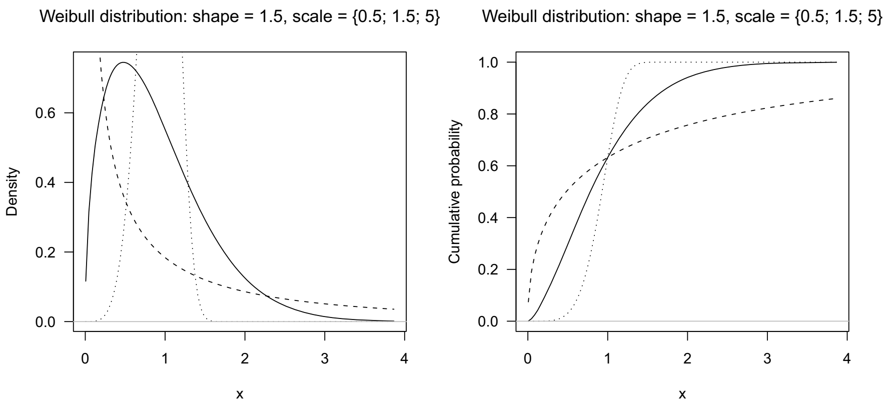


Fig. A.6: The Weibull distribution for $\lambda = 1$ and $\nu = \{0.5; 1.5; 5\}$ (dashed, full, dotted line, respectively)

eters μ and σ^2, where μ and σ^2 are, respectively, the mean and the variance of the population. From the previous sections, we know that the following relationships hold:

$$\mu = \mathrm{E}[X], \qquad \sigma^2 = \mathrm{E}[X^2] - \mathrm{E}[X]^2.$$

Now the *sample estimators* of $\mathrm{E}[X]$ and $\mathrm{E}[X^2]$ are, respectively,

$$\frac{1}{n}\sum_{i=1}^{n} X_i = \bar{X} \qquad \text{and} \qquad \frac{1}{n}\sum_{i=1}^{n} X_i^2;$$

therefore the moment estimators of μ and σ^2 are

$$\hat{\mu} = \bar{X} \qquad \text{and} \qquad \hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n} X_i^2 - \bar{X}^2.$$

Now suppose we extract an i.i.d. sample of size n from a population to investigate the phenomenon, say the vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]$. Then, the *(point) estimates* of μ and σ^2 obtained with the moment estimation are:

$$\hat{\mu} = \frac{1}{n} x_i \qquad \text{and} \qquad \hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \bar{x}^2.$$

Note that the estimator is itself an r.v., whereas the estimate is a realization of an r.v., although we will use the same symbols, the context being sufficiently clear.

Other useful sample indicators are the *minimum* and the *maximum*, respectively denoted by the symbols $X_{(1)} = \min_i X_i$ and $X_{(n)} = \max_i X_i$.

There are other ways to obtain a point estimate, to cite but two, the *maximum likelihood* and the *least squares estimation*, which will be discussed later in this appendix.

Distribution of the Most Common Sample Estimators When Observations Are i.i.d.

The estimator of the parameter of interest is an r.v. because its realization depends on the sample given. The estimator of the mean of the distribution is usually the sample mean. This estimator often has the same distribution as the observations in the sample. If the sample is made of independent and identically distributed (i.i.d.) observations, it is easy to obtain the expected value and variance of $\bar{X}$. For instance, if $[X_1, \dots, X_n]$ is a vector of i.i.d. r.v.s from a distribution with mean μ and variance σ^2, then

$$\begin{aligned}
\mathrm{E}[\bar{X}] &= \mathrm{E}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] = \frac{1}{n}\sum_{i=1}^{n} \mathrm{E}[X_i] = \frac{1}{n}\sum_{i=1}^{n} \mu = \mu; \\
\mathrm{V}[\bar{X}] &= \mathrm{V}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] = \frac{1}{n^2}\left[\sum_{i=1}^{n} \mathrm{V}[X_i] + 2\sum_{i=1}^{n}\sum_{j\neq i} \mathrm{COV}(X_i, X_j)\right] \\
&= \frac{1}{n^2}\sum_{i=1}^{n} \sigma^2 = \frac{\sigma^2}{n},
\end{aligned}$$

where $\mathrm{COV}(X_i, X_j)$ is the covariance between X_i and X_j. Note that the last result is due to the assumption of independence among observations, which implies $\mathrm{COV}(X_i, X_j) = 0, i \neq j$.

The above results are valid whenever the sample is made of i.i.d. observations and when the common distribution admits finite expected value and variance. As regards the distribution of $\bar{X}$, it really depends on the distribution of X_i. For instance, if $[X_1, \dots, X_n]$ is a vector of independent r.v.s and $X_i \sim N(\mu_i, \sigma_i^2)$, $i = 1, \dots, n$, then

$$\sum_{i=1}^{n}(a_i + b_i X_i) \sim N\left(\sum_{i=1}^{n}(a_i + b_i\mu_i), \sum_{i=1}^{n} b_i^2\sigma_i^2\right) \qquad a_i, b_i \in \mathbb{R}.$$

As a consequence, if we let $\mu_i = \mu$, $\sigma_i^2 = \sigma^2$ (hence the X_i's are i.i.d.), $a_i = 0$, and $b_i = n^{-1}$ for all i, then:

$$\bar{X} \sim N\left(\mu, \frac{\sigma^2}{n}\right).$$

Another common example is given by considering a sample of i.i.d. observations from a Bernoulli distribution; that is, when $X_i = 1$ has a Bernoulli distribution with parameter θ. Recall from Sect. A.1.2 that $S = \sum_{i=1}^{n} X_i \sim \mathrm{Bi}(n, \theta)$; since $\bar{X} = S/n$, $\bar{X}$ has the same probability function of S, but a different domain. In particular $D_{\bar{X}} = \{0, 1/n, 2/n, \ldots, 1\}$ and $\Pr\{\bar{X} = s/n\} = \Pr\{S = s\}$. This means that, in this case, it is possible to obtain exact inference on the parameter θ (see Sect. A.3).

Given that $\mathrm{E}[X_i] = \theta$ and $\mathrm{V}[X_i] = \theta(1-\theta)$, we have that, from the general results on $\bar{X}$ when observations are i.i.d.,

$$\mathrm{E}[\bar{X}] = \theta \qquad \text{and} \qquad \mathrm{V}[\bar{X}] = \frac{\theta(1-\theta)}{n}.$$

The distribution of $X_{(n)}$, the *maximum* of n i.i.d. random variables distributed as $F_X(x)$, is obtained by realizing that the event $\{X_{(n)} \leq x\}$, $x \in D_X$ implies the event $\cap_{i=1}^n \{X_i \leq x\}$, and by the definition of independent r.v.s. Thus

$$F_{X_{(n)}}(x) = \Pr\{X_{(n)} \leq x\} = \prod_{i=1}^{n} \Pr\{X_i \leq x\} = F_X(x)^n.$$

The distribution of $X_{(1)}$, the *minimum* of n i.i.d. random variables distributed as $F_X(x)$, is obtained by realizing that the event $\{X_{(1)} > x\}$ implies the event $\cap_{i=1}^n \{X_i > x\}$, and by the definition of independent r.v.s. Thus

$$F_{X_{(1)}}(x) = 1 - \Pr\{X_{(1)} > x\} = 1 - \prod_{i=1}^{n} \Pr\{X_i > x\} = 1 - [1 - F_X(x)]^n.$$

The density functions of $X_{(1)}$ and $X_{(n)}$ are, respectively,

$$f_{X_{(1)}}(x) = n f_X(x)[1 - F_X(x)]^{(n-1)} \qquad \text{and} \qquad f_{X_{(n)}}(x) = n f_X(x) F_X(x)^{(n-1)}.$$

Their distributions can be written explicitly only in some cases, for instance, when $X \sim U[a, b]$ or when $X \sim \mathrm{Exp}(\lambda)$.

Properties of "Good" Estimators

Since an estimator is an r.v., it is possible to obtain the expected value and variance. These quantities are very useful when comparing different estimators. A first requirement of an estimator is that, on average, it yields the true value of the parameter of interest. This property is called *unbiasedness*. Formally, if $\hat{\theta}$ is an estimator of the parameter θ, the requirement can be written as

$$\mathrm{E}[\hat{\theta}] = \theta \qquad\qquad \forall\, \theta \in \Theta.$$

This computation can usually be done because of the assumption that the sample observations are i.i.d. from a specified statistical model. For instance, let $X_1, \ldots, X_n$ be a random sample from a normal distribution with parameters μ and σ^2. Then, because the r.v.s are identically distributed, we have $\mathrm{E}[X_i] = \mu$ and $\mathrm{V}[X_i] = \sigma^2$ for all i. For instance, the expected value of the moment estimator of μ is

$$\mathrm{E}[\hat{\mu}] = \mathrm{E}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] = \frac{1}{n}\sum_{i=1}^{n}\mathrm{E}[X_i] = \frac{1}{n}\sum_{i=1}^{n}\mu = \mu,$$

and therefore $\hat{\mu}$ is an unbiased estimator of μ. This is not true for the moment estimator of the variance, since

$$\begin{aligned}\mathrm{E}[\hat{\sigma}^2] &= \mathrm{E}\left[\frac{1}{n}\sum_{i=1}^{n} X_i^2 - \bar{X}^2\right] = \frac{1}{n}\sum_{i=1}^{n}\mathrm{E}[X_i^2] - \mathrm{E}[\bar{X}^2] \\ &= \frac{1}{n}\sum_{i=1}^{n}(\mu^2+\sigma^2) - \left(\mu^2 + \frac{\sigma^2}{n}\right) = \sigma^2\left(\frac{n-1}{n}\right).\end{aligned}$$

In the last equation, we have used the relationships

$$\begin{aligned}\mathrm{V}[X_i] = \mathrm{E}[X_i^2] - \mathrm{E}[X_i]^2 \quad &\Rightarrow \quad \mathrm{E}[X_i^2] = \mathrm{V}[X_i] + \mathrm{E}[X_i]^2 = \sigma^2 + \mu^2; \\ \mathrm{V}[\bar{X}] = \mathrm{E}[\bar{X}^2] - \mathrm{E}[\bar{X}]^2 \quad &\Rightarrow \quad \mathrm{E}[\bar{X}^2] = \mathrm{V}[\bar{X}] + \mathrm{E}[\bar{X}]^2 = \frac{\sigma^2}{n} + \mu^2.\end{aligned}$$

The expectation of $\hat{\sigma}^2$ tells us that, on average, the estimator $\hat{\sigma}^2$ underestimates the true value of the parameter σ^2, and therefore $\hat{\sigma}^2$ is said to be *biased*. However, note that

$$\lim_{n\to+\infty}\mathrm{E}[\hat{\sigma}^2] = \sigma^2,$$

and $\hat{\sigma}^2$ is then asymptotically unbiased. Usually, statisticians prefer to consider the following unbiased estimator of σ^2:

$$s^2 = \left(\frac{n}{n-1}\right)\hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}[X_i - \bar{X}]^2.$$

Another important property of an estimator is *consistency*. Formally, the (weak) consistency of an estimator requires that the estimator converges in probability to the true value of the parameter of interest. That is, given an estimator $\hat{\theta}_n$ (that depends on the sample size n),

$$\lim_{n\to+\infty}\Pr\left\{\,|\hat{\theta}_n - \theta| > \epsilon\right\} = 0 \qquad\qquad \forall\, \epsilon > 0.$$

Roughly speaking, if the amount of information increases, then we expect the estimator to be distributed around the true value of the parameter, and the accuracy should increase with n (i.e., the variance of the distribution of $\hat{\theta}$ should decrease with n). Two sufficient conditions to ensure that an estimator is (weakly) consistent are

$$\mathrm{E}[\hat{\theta}] = \theta \qquad \text{and} \qquad \lim_{n\to+\infty}\mathrm{V}[\hat{\theta}] = 0.$$

For instance, the estimator of the mean of an i.i.d. sample $X_i \sim N(\mu,\sigma^2)$, $i = 1,\ldots,n$ is weakly consistent since $\mathrm{E}[\hat{\mu}] = \mu$, and $\mathrm{V}[\hat{\mu}] = \sigma^2/n$.

Central Limit Theorem

A sequence $Z_1, \ldots, Z_n$ of r.v.s with distribution functions $F_{Z_1}(t), \ldots, F_{Z_n}(t)$ converges in distribution to an r.v. Y with distribution function $F_Y(y)$, written $Z_n \xrightarrow{d} Y$, if

$$\lim_{n \to +\infty} F_{Z_n}(t) = F_Y(t) \qquad t \in I\!R.$$

We can then state the central limit theorem as follows.

Let $[X_1, X_2, \ldots, X_n]$ be n i.i.d. r.v.s from a common distribution $F_X(x)$, with finite first and second moment, and let $\bar{X}_n = n^{-1} \sum_{i=1}^{n} X_i$. Then, as n increases, we have

$$Z_n = \frac{\sqrt{n}(\bar{X}_n - \mathrm{E}[X_i])}{\sqrt{\mathrm{V}(X_i)}} \quad \xrightarrow{d} \quad N(0,1).$$

This theorem is important because it allows us to obtain the distribution of some test statistics depending on the mean of n i.i.d. r.v.s.

A.3 Hypothesis Testing

Let us go back to the explanatory example of Sect. A.1. Suppose that we are interested in evaluating whether the probability of an algorithm to find a solution within a certain time limit t_0 is more than or equal to 90%. To do that, suppose that we run the algorithm $1,000$ times, and find out that the runtime is less than t_0 874 times. A point estimation of the parameter θ gives $\hat{\theta} = 0.874$. This value is less than 0.9, but is it sufficiently far from 0.9 to be sure enough that our conjecture cannot be realistic? What would change if we had run the algorithm $n = 100$ times and found that the runtime is less than t_0 87 times? Would our conclusion be the same?

Null and Alternative Hypotheses

The question "is the probability of the algorithm to have a runtime less than t_0 more than or equal to 90%?" is called the *null hypothesis*. There is an *alternative* hypothesis which can be true, i.e., that the probability of the algorithm having a runtime less than t_0 is less than 90%.

Since our decision must be made on the available information (that of the sample), it is impossible to answer the question with no margin of error. The theory of hypothesis testing was born in order to answer these questions, bounding and quantifying the probability of incorrectly rejecting the null hypothesis.

In the previous paragraphs we saw that the estimator of a parameter is an r.v. and that its distribution is described by a probability law that depends on some parameters of the population. How would this probability distribution look if the null hypothesis were true? Is the probability of observing the estimate of θ given by the sample "too small" to trust that the null hypothesis is true? Or in our previous example, is $\hat{\theta}$ "too far" from 0.9 to decide that the null hypothesis should be rejected?

The null and alternative hypotheses can be formulated as follows:

$$\begin{cases} H_0 : \theta \in \Theta_0 \\ H_1 : \theta \in \Theta_1 \end{cases},$$

where $\Theta_0 \cup \Theta_1 = \Theta$ and $\Theta_0 \cap \Theta_1 = \emptyset$. The subset Θ_0 specifies the values of the parameter which are in agreement with the null hypothesis. In our previous example $\Theta_0 = [0.9, 1]$, and $\Theta_1 = [0, 0.9)$. Note that in the null hypothesis there is always a well-specified value of the parameter θ (e.g., the value 0.9 belongs to Θ_0; this will be clearer in what follows).

Statistical Test and Acceptance/Rejection Region in the Sample Space

A *statistical test* is a partition of the *sample space* $\mathcal{X}$, where the sample space is the set of all possible values of the random vector of sample data $\mathbf{X}$. In other words, there are some points of the sample space $\mathbf{x} \in \mathcal{X}_0 \subset \mathcal{X}$ which are in agreement with the null hypothesis, and others $\mathbf{x} \in \mathcal{X}_1 \subset \mathcal{X}$ which are too unlikely to assume that the null hypothesis holds. Thus, the observed data may lead to the *rejection* of the null hypothesis or not. The information of the data is summarized by the *test statistic* $T = T(\mathbf{X})$, which is a function of the random vector of data $\mathbf{X}$ whose probability distribution is known if the null hypothesis is true. Given that the distribution of the test statistic is known under the null hypothesis, we may take a decision based on $T(\mathbf{x})$, the value of the test statistic computed with the vector of observed data $\mathbf{x}$.

The domain of $T(\mathbf{X})$ can be partitioned into an *acceptance region* $A(\mathbf{X})$ and a *rejection region* $R(\mathbf{X})$ of the null hypothesis.

Once the sample data $\mathbf{x}$ have been observed, we may conclude that

$$\begin{cases} \text{we cannot reject } H_0 & \text{if } T(\mathbf{x}) \in A(\mathbf{X}) \\ \text{we reject } H_0 & \text{if } T(\mathbf{x}) \in R(\mathbf{X}). \end{cases}$$

Type I and Type II Errors

The acceptance or rejection of H_0 is therefore induced by the sample data $\mathbf{x}$ through the test statistic T. Hence, there are two kinds of errors that may arise, which are known as type I (α) and type II (β) errors:

$$\alpha = \Pr_{\theta \in \Theta_0} \{T(X) \in R(\mathbf{X})\}$$
$$\beta = \Pr_{\theta \in \Theta_1} \{T(X) \in A(\mathbf{X})\}$$

That is, α is the probability of incorrectly rejecting H_0 when H_0 is true; β is the probability of not rejecting H_0 when the alternative hypothesis H_1 is true. A theoretical "perfect" test should satisfy $\alpha = \beta = 0$. In practice, the variability of the sample data vector $\mathbf{X}$ cannot ensure this ideal condition. Operatively, it is impossible to control both kinds of error, because the true value of θ is unknown (especially under the alternative hypothesis).

The role of inferential statistics is thus to try to control at least one of these errors. The error that we can control is α, as there is always a well-specified value of θ in Θ_0. Indeed, the well-specified value of θ in Θ_0 typically maximizes the probability of a type I error. Thus, the rejection region $R(\mathbf{X})$ is determined for an a priori chosen α that satisfies the condition

$$\alpha = \sup_{\theta \in \Theta_0} \Pr_{\theta}\{T(X) \in R(\mathbf{X})\}, \tag{A.1}$$

which is also known as the *significance level* of the test.[3] We will better explain this last sentence by referring to the introductory example of this section.

In Sect. A.2, we have seen that $n\hat{\theta} = \sum_{i=1}^{n} X_i \sim \mathrm{Bi}(n, \theta)$ when the X_i's are i.i.d. r.v.s with a Bernoulli distribution and $\Pr\{X_i = 1\} = \theta$. Now recall the null hypothesis $H_0 : \theta \geq 0.9$ of the example. This means that, if H_0 is true, θ belongs to the (closed) interval $[0.9, 1]$. There are infinitely many points in this interval, each specifying a probability distribution of the r.v. $n\hat{\theta}$ under H_0. Thus, in order to compute the probability of making a type I error, we have to specify the rejection region of the test $R(\mathbf{X})$ first. This region will be made of the points $\mathbf{x} \in \mathcal{X}$ for which $\hat{\theta}$ is "too small" with respect to the border value $\theta = 0.9$ of Θ_0; that is, $R(\mathbf{X})$ will be an interval that satisfies

$$R(\mathbf{X}) = \{\mathbf{x} \in \mathcal{X} : n\hat{\theta} \leq c(\alpha)\},$$

for a constant $c(\alpha)$ to be specified, known as the *critical value* of the test. Now let $T(X) = n\hat{\theta}$ be the test statistic, whose distribution is $\mathrm{Bi}(n, \theta)$, $\theta \in \Theta_0$, when the null hypothesis is true. For any value of $\theta \in \Theta_0$ we can obtain a constant $c(\alpha)$ satisfying the above condition on $R(\mathbf{X})$. Given that the rejection region has the form $[0, c(\alpha)]$, and given that if $\theta_1 \leq \theta_2$ are two points of Θ_0

$$\Pr_{\theta_1}\{n\hat{\theta} \leq c\} \geq \Pr_{\theta_2}\{n\hat{\theta} \leq c\},$$

the value of $\theta \in \Theta_0$ that maximizes the type I error is the boundary point $\theta = 0.9$; that is

$$\alpha = \sup_{\theta \in \Theta_0} \Pr_{\theta}\{T(X) \in R(\mathbf{X})\} = \Pr_{\theta}\{T(X) \in R(\mathbf{X})\}_{|\theta = 0.9}.$$

The above probability is known as the significance level of the test, and the rejection region $R(\mathbf{X})$ will then be specified by choosing a desired α-level $\in (0, 1)$ and by focusing attention on the case when $n\hat{\theta} \sim \mathrm{Bi}(n, 0.9)$ and $n = 1{,}000$. In other words, we will base the inference on θ on the Binomial model with parameters $n = 1{,}000$

[3] Note that we have used the same symbol α to specify both the type I error and the significance level of the test. This is because, when the null hypothesis is of the kind $H_0 : \theta = \theta_0$ (i.e., the null parameter space consists in only one point), the two definitions coincide.

and $\theta = 0.9$, because the other elements of Θ_0 would lead to a smaller type I error. The probabilistic model maximizing the type I error is known as the *null distribution* of the test statistic.

The significance level α indicates how much we are willing to risk (in terms of probability) an incorrect rejection of H_0 when the latter is true. Some typical choices of α are 1%, 5%, and 10%, although the significance level of the test is a subjective choice (and the p-value approach described in next paragraph will reduce the role of α). To fix the ideas, suppose that we choose $\alpha = 5\%$: The discrete nature of the binomial distribution does not allows us to find a quantile which satisfies $\Pr_{\theta=0.9}\{n\hat{\theta} < c(\alpha)\} = \alpha$ exactly. We can thus choose the quantile of the null distribution whose cdf is closest to the chosen α-level of the test (or not bigger than α). By looking at the quantiles of the binomial distribution with parameters $n = 1,000$ and $\theta = 0.9$, we find that

$$\Pr_{\theta=0.9}\{1000 \cdot \hat{\theta} \leq 883\} = 0.0433, \qquad \text{and} \qquad \Pr_{\theta=0.9}\{1000 \cdot \hat{\theta} \leq 884\} = 0.0534.$$

Thus, we can set the critical value equal to $c(\alpha') = 883$, and perform the test with a significance level which is actually equal to $\alpha' = 4.33\%$. Therefore, the rejection and acceptance regions of the test will be

$$R(\mathbf{X}) = \{\mathbf{x} \in \mathcal{X} : 1000 \cdot \hat{\theta} \leq 883\} \qquad \text{and} \qquad A(\mathbf{X}) = \{\mathbf{x} \in \mathcal{X} : 1000 \cdot \hat{\theta} \geq 884\}.$$

In our example, $1,000 \cdot \hat{\theta} = 874$, which belongs to $R(\mathbf{X})$; then, we will reject the null hypothesis at a significance level $\alpha' = 4.33\%$.[4]

Note that the null sample space $\mathcal{X}_0$ is identified by the points of $A(\mathbf{X})$ and vice versa; that is:

$$\mathbf{x} \in \mathcal{X}_0 \qquad \Longleftrightarrow \qquad T(\mathbf{x}) \in A(\mathbf{X}).$$

There is another way to solve the testing problem above. Given that in our example n is very large, we could have applied the central limit theorem in order to specify the null distribution of a different test statistic $T_n(\mathbf{X})$. If X_i $i = 1, \ldots, n$ are i.i.d. dichotomous variables with $\theta = \Pr\{X_i = 1\}$, then we known that $\mathrm{E}[X_i] = \theta$ and $\mathrm{V}[X_i] = \theta(1-\theta)$. Thus, if we let

$$T_n(\mathbf{X}; \theta) = \sqrt{n}\frac{(\hat{\theta} - \theta)}{\sqrt{\theta(1-\theta)}},$$

we know by the central limit theorem that $T_n(\mathbf{X}; \theta)$ is approximately distributed as a standard normal r.v., if θ is the true value of the parameter. Now under the null hypothesis, $\theta \geq 0.9$ and note that $\{T_n(\mathbf{X}; \theta) \geq 0\}$ implies $\{\hat{\theta} \geq \theta\}$. This means that, if we set $\theta = 0.9$, positive values of $T_n(\mathbf{X}; \theta)$ are in accordance with the null

[4] It is worth noting that, if the observed value of the test statistic belongs to $A(\mathbf{X})$, this does not mean that we have a further knowledge about the true value of the parameter θ in the population. We can only conclude that the observed data do not disagree "enough" with H_0 to reject it at the specified significance level.

hypothesis, whereas negative values of $T_n(\mathbf{X};\theta)$ are in disagreement with the null hypothesis. Hence, the rejection region $R(\mathbf{X})$ will be of the form $(-\infty, c(\alpha)]$, with $c(\alpha)$ being the critical value of the test. It is easy to see that, given two points of Θ_0, say $\theta_1 \leq \theta_2$, $\Pr_{\theta_1}\{T_n(\mathbf{X};\theta_1) \leq c(\alpha)\} \geq \Pr_{\theta_2}\{T_n(\mathbf{X};\theta_1) \leq c(\alpha)\}$, again the boundary point $\theta = 0.9$ maximizes the type I error for any given $c(\alpha)$. Therefore we have that

$$T_{1000}(\mathbf{x};\theta = 0.9) = \sqrt{1000}\frac{.874 - 0.9}{\sqrt{0.9 \cdot 0.1}} = -2.740641$$

is now the observed value of the test statistic, and that this value can be compared with the quantiles of the limiting distribution of $T_n(\mathbf{X})$ (that is, the standard normal) in order to determine whether $T(\mathbf{x};\theta = 0.9)$ falls into the rejection region of the null hypothesis or not. Let $\alpha = 5\%$; the quantile $z_{0.05}$ of the standard normal distribution is equal to -1.6448, therefore

$$R(\mathbf{X}) = \{\mathbf{x} \in \mathcal{X} : T(\mathbf{x};\theta = 0.9) \leq -1.6448\},$$
$$A(\mathbf{X}) = \{\mathbf{x} \in \mathcal{X} : T(\mathbf{x};\theta = 0.9) > -1.6448\}.$$

Since the observed value of the test statistic falls into the rejection region, we will reject the null hypothesis at a 5% significance level.

The two testing approaches introduced in this paragraph lead to the same conclusion. It is worth noting that they are indeed slightly different: in the first approach we have compared the observed value of the test statistics $n\hat{\theta}$ with the quantiles of its theoretical distribution evaluated the boundary point $\theta = 0.9$, whereas in the second approach we have compared the value $T_n(\mathbf{x})$ with its asymptotic distribution, which is a standard normal as the sample size n tends to infinity. The first test is said to be an *exact* test, whereas the second is said to be *asymptotic*. The difference between the two approaches tends to vanish as n tends to infinity, but may not be negligible for small n, as we will show in the next paragraph.

As a final remark, the conclusions of hypothesis testing are conditioned by the amount of information available. To see this, suppose $n = 100$ and $\hat{\theta} = 0.87$; by applying both the exact and asymptotic tests one would not reject the null hypothesis $H_0 : \theta \geq 0.9$ at a significance level of $\alpha = 5\%$. This happens because, when the available amount of information increases, a "good" test should better discriminate between the null and alternative hypotheses. This property is known as the *consistency* of a test, and it may not hold for some tests.

The p-Value Approach

The acceptance–rejection method of testing hypotheses that we have just introduced is unable to capture all the latent information in the data. For instance, if we had repeated the experiment and found that in 870 cases the runtime did not exceed t_0, we would have rejected the null hypothesis $H_0 : \theta \geq 0.9$ as well. However, clearly an estimate of θ equal to 0.870 is slightly smaller than the previous one, which was

$\hat{\theta} = 0.874$. It would be better to have an idea of how far the observed data are from the boundary point of the null parameter space Θ_0, for instance. This can be done by computing the *observed significance level* (or p-value), which is defined as *the minimum significance level for which the null hypothesis would be rejected.* Formally, the p-value is defined as

$$p\text{-value} = \min_{\alpha} \sup_{\theta \in \Theta_0} \Pr_{\theta}\{\mathbf{X} \in R(\mathbf{X}; \alpha)\},$$

where the emphasis is on the fact that the rejection region is specified by α. Going back to our previous example (the exact testing approach), we have that

$$\Pr_{\theta=.9}\{T(\mathbf{X}) \leq 874\} = 0.0045,$$

so, with the observed data, we should have chosen a significance level $\alpha = 0.45\%$ in order to reject the null hypothesis.

The p-value is much more informative about the rejection of the null hypothesis than the acceptance–rejection approach because, ceteris paribus, we could have chosen a significance level about ten times smaller than 4.33% and rejected the null hypothesis as well.

According to the asymptotic approach, the p-value is equal to $\Phi(-2.740641) = 0.0031$, so now the difference between the exact and asymptotic approaches becomes more evident. In both cases there is strong evidence against the null hypothesis. Of course, for fixed α, there is the equivalence

$$p\text{-value} < \alpha \qquad \Longleftrightarrow \qquad T(\mathbf{X}) \in R(\mathbf{X}; \alpha).$$

The example above is a test with one-sided alternative (i.e., when H_1 is of the form $\theta < \theta_0$ or $\theta > \theta_0$). There are also tests with two-sided alternatives, when $H_0 : \theta = \theta_0$ and $H_1 : \theta \neq \theta_0$. In this kind of testing problem, the null hypothesis should be rejected whenever $|\hat{\theta} - \theta_0|$ is "too big." In this case, the acceptance and rejection regions are of the form

$$\begin{aligned} A(\mathbf{X}) &= \{\mathbf{x} \in \mathcal{X} : c_1(\alpha) \leq T(\mathbf{X}) \leq c_2(\alpha)\}, \\ R(\mathbf{X}) &= \{\mathbf{x} \in \mathcal{X} : T(\mathbf{X}) < c_1(\alpha) \, \cup \, T(\mathbf{X}) > c_2(\alpha)\}; \end{aligned}$$

with α being the significance level of the test. Usually, $c_1(\alpha)$ and $c_2(\alpha)$ are determined in order to be $\alpha/2 = \Pr_{\theta_0}\{T(\mathbf{X}) < c_1(\alpha)\} = \Pr_{\theta_0}\{T(\mathbf{X}) > c_2(\alpha)\}$.

The p-value computation, in this case, is

$$p\text{-value} = 2 \min \left\{ \Pr_{\theta_0}\{T(\mathbf{X}) \leq T(\mathbf{x})\}, \Pr_{\theta_0}\{T(\mathbf{X}) \geq T(\mathbf{x}))\} \right\}.$$

For instance, if we apply an exact test assessing $H_0 : \theta = 0.9$, then $\Pr_{\theta_0}\{T(\mathbf{X}) \leq 874) = 0.0045$, $\Pr_{\theta_0}\{T(\mathbf{X}) \geq 874) = 0.9966$, therefore the p-value is equal to $2 \times 0.0045 = 0.009$.

Brief Introduction to Permutation Tests

In the previous paragraphs, we have introduced two examples of *parametric* statistical tests. The parametric approach requires that the statistical model generating the observed data be known; moreover, the null distribution of the test statistic is often approximated, and there are several examples where the conditions (*regularity conditions*) that ensure the properties of the *likelihood ratio test* are not met. Permutation tests are a further approach that removes some of these issues. It has been becoming more popular in the recent years thanks to the advent of fast computers, although the theory behind it can be traced back to Fisher in the 1930s.

Permutation tests are exact testing procedures that do not require assumptions on the probability distribution of data. They are based on the notion of *exchangeability* of data. The r.v.s $X_1, X_2, \dots, X_n$ are said to be exchangeable if their joint distribution is equal to the joint distribution of a permutation of $X_1, X_2, \dots, X_n$. For instance, n i.i.d. r.v.s are always exchangeable, because their joint distribution can be written as the product of their densities/probability functions (because of the commutative property of the product operator). Let $\mathbf{X} = [X_1, X_2, \dots, X_n]$ be an n-dimensional vector, then $X_1, X_2, \dots, X_n$ are exchangeable if

$$\Pr\{\mathbf{X}\} = \Pr\{X_1, X_2, \dots, X_n\} = \Pr\{X_{\pi_1}, X_{\pi_2}, \dots, X_{\pi_n}\} = \Pr\{\mathbf{X}^*\},$$

where $\pi_1, \pi_2, \dots, \pi_n$ is a random permutation of the first n integers.

Suppose that we want to compare two algorithms on n instances. At each run we record the dichotomous variable X_i $i = 1, \dots, n$, where $X_i = 1$ means "algorithm A has smaller runtime than algorithm B," and $X_i = 0$ meaning the opposite. We want to test the null hypothesis that the two algorithms have the same performance against the alternative that algorithm A is better. This can be translated into $H_0 : \theta = 0.5$ versus $H_1 : \theta > 0.5$. The sample space of the experiment is made of 2^n points. Since the test is a partition of the sample space $\mathcal{X}$, one can obtain the exact distribution of the test statistics by computing the value of a test statistic at all points of $\mathcal{X}$. This is equivalent to the exact parametric testing approach to this problem, i.e., if the test statistic is $n\hat{\theta} = \sum_{i=1}^{n} X_i$, the resulting probability distribution is again binomial with parameters n and $\theta = 0.5$.

To see this note that, if $n = 5$ and $\mathbf{x} = [0, 0, 1, 1, 0]$, there are $\binom{5}{2} = 10$ permutations of the vector $\mathbf{x}$ that lead to the same estimate of $\hat{\theta} = 0.2$ over $2^5 = 32$ possible permutations of $\mathbf{x}$. Therefore, the probability of observing $\hat{\theta} = 0.2$ in the sample space is $10/32 = 0.3125$. The binomial model introduced in the previous paragraphs would give us the same result: the probability of the event $X = 2$, when $X \sim \mathrm{Bi}(5, 0.5)$ is

$$\binom{5}{2} 0.5^2 (1 - 0.5)^3 = 10 \cdot 0.03125 = 0.3125.$$

If, in addition, the runtimes of algorithms A and B are also recorded (see Table A.1), then we are dealing with a continuous variable which can be modeled as

	1	2	3	4	5
X_A	0.591	1.587	0.210	0.158	0.797
X_B	0.490	0.315	0.641	1.413	0.401
X	0	0	1	1	0

Table A.1: Runtime results (in seconds) of algorithms A (X_A) and B (X_B) in $n = 5$ instances. The event $X = 1$ means "A is faster than B"

$X_{iA} = \mu + \delta + \varepsilon_{iA}$ and $X_{iB} = \mu + \varepsilon_{iB}$. Let us assume that the r.v.s ε_{iA} and ε_{iB} are i.i.d. Under this assumption, the random variable X_{iA} satisfies $X_{iA} \overset{d}{<} X_{iB}$ (i.e., algorithm A is faster than B) only if $\delta < 0$. Note that we have only assumed the r.v.s ε_{iA} and ε_{iB} to be i.i.d. (it would be sufficient that they are exchangeable, not necessarily i.i.d.). The null hypothesis of equal runtime performances can be expressed by $H_0 : \delta = 0$; note that under H_0 we have $X_{iA} \overset{d}{=} X_{iB}$, i.e., the random variables X_{iA} and X_{iB} have identical distribution. This means that, if H_0 is true, the observed data $\mathbf{x} = [x_{1A}, \dots, x_{nA}, x_{1B}, \dots, x_{nB}]$ are independent realizations of the same r.v. If this is true, the probability of observing $\mathbf{x}$ is the same as that of observing $\mathbf{x}^*$, where $\mathbf{x}^*$ is a random permutation of $\mathbf{x}$. In other words, the data of algorithm A could have been generated from X_{iB} and vice versa. Thus, in order to perform a permutation test to assess $H_0 : \delta = 0$ against $H_1 : \delta < 0$ we consider all $\binom{2n}{n}$ possible permutations of $\mathbf{x}$, choose a suitable test statistics (for instance, the difference of the means $T^* = T(\mathbf{x}^*) = \bar{x}^*_A - \bar{x}^*_B$, and obtain its null distribution by computing the value of T^* for any (distinct) random permutation of $\mathbf{x}$. The observed value of the test statistic $T = T(\mathbf{x}) = \bar{x}_A - \bar{x}_B$ (i.e., the value of the test statistic obtained from the observed data) will then be compared with the null (permutation) distribution of T^* in order to compute a p-value. Note that, in this example, small (negative) values of T are significant against the null hypothesis.

Formally, let $T^*_{(1)} \leq T^*_{(2)} \leq \dots, \leq T^*_{(M)}$ be the values of T^* computed at each point $\mathbf{x}^* : \mathbf{x}^* = \pi(\mathbf{x}), \pi \in \Pi$, where Π is the set of all permutations of the first $2n$ natural integers and N is its cardinality. Let $T^*_{[N\alpha]}$ be the α-quantile of the permutation distribution. Then $H_0 : \delta = 0$ will be rejected in favor of the alternative $H_1 : \delta < 0$ at a significance level α if $T \leq T^*_{[N\alpha]}$. Note that this is equivalent to obtaining a p-value from the null distribution and comparing it with the nominal significance level α. We can define the p-value of this example as

$$p = \hat{F}_{T^*}(T) = \frac{1}{N}\sum_{b=1}^{N} I(T^*_{(b)} \leq T).$$

Thus, the exact p-value in case that the runtime results are recorded as Bernoulli variables (last row of Table A.1), will be equal to: $p = \Pr_{\theta=0.5}\{5 \cdot \hat{\theta} \leq 2\} = 0.5$.

If the observed runtimes are as in the first two rows of Table A.1, the observed value of the test statistic is $T = 0.0166$; Now there are $N' = 10!$ possible distinct permutations of the whole vector of data. If we compute the test statistics for each permutation, we will realize that there are many repetitions. For instance, if we separately permute the elements of $\mathbf{x}_A$ and $\mathbf{x}_B$ in all possible ways, we will obtain the same value of the test statistic $(5!)^2$ times. Thus the number of really informative permutations is in fact $N = \binom{10}{5}$, i.e., all the possible distinct combinations of ten elements in groups of five. The null distribution of T^* can be obtained as follows. Let $\mathbf{x}^*$ be a random permutation of $\mathbf{x}$, consider the first $n_A = 5$ elements of $\mathbf{x}^*$ to be the results of algorithm A, and the last $n_B = 5$ elements of $\mathbf{x}^*$ to be the results of algorithm B, and compute the value of the test statistic $T^* = \bar{x}_A^* - \bar{x}_B^*$, where $\bar{x}_A^* = \sum_{i=1}^n x_{iA}^*/n_A$ and $\bar{x}_B^* = \sum_{i=1}^n x_{iB}^*/n_B$. Repeat this procedure for all, N informative permutations. From the null distribution we can then obtain the p-value, which is equal to 0.5238. Note that this p-value is not necessarily equal to the previous one (obtained with data coded as 0/1), since the support of the test statistic now is made of $N = 252$ points instead of 32. Nevertheless, the conclusion is the same: there is no evidence in the observed data that the null hypothesis should be rejected. What has changed is the amount of information available (the runtimes).

Permutation tests are *conditional* procedures, where conditioning is with respect to the *permutation sample space* $\mathcal{X}^* = \{\mathbf{x}^* : \mathbf{x}^* = \pi(\mathbf{x}), \pi \in \Pi\}$. That is, the sample space is built on the observed vector of data $\mathbf{x}$ and it is induced by the null hypothesis. The distribution function $\hat{F}_{T^*}(t)$, $t \in \mathbb{R}$, is the exact conditional distribution of T^* on $\mathcal{X}^*$. When n is large, it might be impossible to perform all possible N informative permutations: in such a case $\hat{F}_{T^*}(t)$ can be approximated by considering a large number $B < N$ of random permutations of $\mathbf{x}$.

Note that the test statistic we have applied is *permutationally equivalent* to $T^{*\prime} = \bar{x}_A^*$, in the sense that $T^{*\prime}$ leads to the same inferential conclusions (i.e., to the same p-value). This is because, conditionally on $\mathbf{x}$, the mean of the whole vector of data is a constant and, for any permutation $\pi \in \Pi$, there is the relationship $2n\bar{x} = (n_A\bar{x}_A^* + n_B\bar{x}_B^*)$, so $T(\mathbf{x}^*) = \bar{x}_A^*(1 + n_A/n_B) - n\bar{x}/n_B$. Since n_A, n_B, and $\bar{x}$ are constants, $T^* \overset{\pi}{\sim} T^{*\prime}$, where the symbol $\overset{\pi}{\sim}$ means "*is permutationally equivalent to*". It can also be shown that there is no need to standardize the test statistic as we usually do in the parametric framework.

Finally, given the data collected, the minimum possible significance level is the inverse of the cardinality of informative permutations, i.e., $\min_{\mathbf{x}^* \in \mathcal{X}^*}(p\text{-value}) = 1/N$. In our example, with the Bernoulli variable X, $\min_{\mathbf{x}^* \in \mathcal{X}^*}(p\text{-value}) = 1/32$, and if we consider the runtimes, $\min_{\mathbf{x}^* \in \mathcal{X}^*}(p\text{-value}) = 1/252$.

Finally, the p-value of the parametric two-sample t-test is 0.5197. However, the t-test assumes that data are normally distributed, and this is not the case here since there cannot be negative runtimes.

A.4 Confidence Intervals

The result of point estimation, discussed in Sect. A.2, depends on the observed sample. If we repeat the experiment, the resulting estimate of the parameter will differ, because the data will be different (if X is continuous, the probability of observing the same data set is zero). Therefore, it is better to provide an interval of possible values for the unknown parameter θ, rather than a single value of the estimate. The construction of the confidence intervals is based on the *pivotal quantity*, i.e., a statistic that depends on the observed data and on the unknown parameter, and whose probability distribution does not depend on θ. For instance, recall that, by the central limit theorem,

$$T(\mathbf{X};\theta) = \sqrt{n}\frac{\hat{\theta}-\theta}{\sqrt{\theta(1-\theta)}} \qquad \xrightarrow{d} \qquad N(0,1),$$

where θ is the true value of the parameter as n increases. Then $T(\mathbf{X};\theta)$ is a pivotal quantity since its (asymptotic) distribution is standard normal, not depending on θ. Thus, we can define a *random interval* $C(\mathbf{X}) = [c_1(\alpha), c_2(\alpha)]$ such that

$$\Pr\{T(\mathbf{X};\theta) \in C(\mathbf{X})\} = 1-\alpha,$$

which only depends on α and not on θ. Now, by the applying the inverse function $T^{-1}(\cdot)$ with respect to θ, we may write the probability above as

$$\Pr\{B(\mathbf{X}) \ni \theta\} = 1-\alpha,$$

where $B(\mathbf{X})$ is equal to $T^{-1}(C(\mathbf{X}))$. The probability $1-\alpha$ is called the *confidence level*, and it represents how much we trust that the true value of the parameter is contained in the interval $B(\mathbf{X})$ before the experiment takes place. Note that this is an a priori probability, concerning the random interval $C(\mathbf{X})$. Once the data have been collected we obtain the realization of the r.v. $C(\mathbf{x}) = [c_1(\mathbf{x};\alpha), c_2(\mathbf{x};\alpha)]$. Now $C(\mathbf{x})$ is no longer a random interval, so it does not make sense to write "the probability of θ being included in $B(\mathbf{x})$ is $1-\alpha$." But if we could repeat the same experiment (i.e., sampling data from the same population) a large number of times, then we would find that $(1-\alpha)\%$ of the times the interval $B(\mathbf{x})$ contains the true value of the parameter θ. Thus $B(\mathbf{x})$ will contain or not the true value of θ with probability 1 (and we do not know whether this is happening or not), but we have a *confidence* of $(1-\alpha)\%$ that θ is included in $B(\mathbf{X})$.

From the formulas above we derive that

$$\Pr\left\{c_1(\alpha) \le \sqrt{n}\frac{\hat{\theta}-\theta}{\sqrt{\theta(1-\theta)}} \le c_2(\alpha)\right\} = 1-\alpha.$$

This means that, a priori (and if n is large enough for the CLT to take effect)

$$\Pr\left\{\hat{\theta} - c_2(\alpha)\sqrt{\frac{\theta(1-\theta)}{n}} \leq \theta \leq \hat{\theta} - c_1(\alpha)\sqrt{\frac{\theta(1-\theta)}{n}}\right\} = 1 - \alpha.$$

Since $B(\mathbf{x})$ must not depend on the unknown parameter θ, it will be evaluated by plugging in the estimate of θ. Thus

$$B(\mathbf{x}) = \left[\hat{\theta} - c_2(\alpha)\sqrt{\frac{\hat{\theta}(1-\hat{\theta})}{n}}, \hat{\theta} - c_1(\alpha)\sqrt{\frac{\hat{\theta}(1-\hat{\theta})}{n}}\right]$$

is an approximate (i.e., asymptotic) confidence interval of level $1-\alpha$. We only need to fix the constants $c_1(\alpha)$ and $c_2(\alpha)$ in order to obtain the confidence interval for θ. In practice, they are determined by letting

$$\Pr\{T(\mathbf{X};\theta) \leq c_1(\alpha)\} = \Pr\{T(\mathbf{X};\theta) \geq c_2(\alpha)\} = \alpha/2.$$

For instance, if we choose a confidence level of 95%, then $c_2(\mathbf{X};\alpha) = -c_1(\mathbf{X};\alpha) = 1.96$.

Going back to our example in the previous section, where we were trying to establish the probability θ for an algorithm to solve an instance within a time limit of t_0, the (approximate) confidence interval for θ with confidence level equal to 95% is given by:

$$\left[0.874 - 1.96\sqrt{\frac{0.874\cdot(1-0.874)}{1000}}, 0.874 + 1.96\sqrt{\frac{0.874\cdot(1-0.874)}{1000}}\right] = [0.853, 0.894]$$

Note that the confidence intervals are always "centered" on the estimate of the parameter of interest.

It can be shown that there is a one-to-one correspondence between (two-sided) statistical tests and confidence intervals.

Therefore, a quick way to test for two-sided alternative hypotheses at a significance level α is choosing the desired confidence level $1-\alpha$, obtaining the corresponding confidence interval, and checking whether the null value of the parameter θ_0 is included in the interval or not. Of course, one can construct "one-sided" intervals, in order to perform a one-sided test.

Returning to our example, we could build a confidence interval on the parameter θ in order to test for $H_0 : \theta \geq 0.9$ in the following form:

$$1 - \alpha = \Pr\{T(\mathbf{X};\theta) \geq c(\alpha)\}.$$

where $c(\alpha)$ should be the α-quantile of a standard normal distribution. Then

$$1-\alpha = \Pr\left\{\sqrt{n}\frac{\hat{\theta}-\theta}{\sqrt{\theta(1-\theta)}} \geq c(\alpha)\right\} = \Pr\left\{\theta \leq \hat{\theta} - c(\alpha)\sqrt{\frac{\theta(1-\theta)}{n}}\right\},$$

If we choose $1-\alpha = 95\%$, then $c(\alpha) = -1.6448$. As before, we plug in the estimate of θ and obtain the confidence interval

$$\text{C.I.} = \left[0, \hat{\theta} - c(\alpha)\sqrt{\frac{\hat{\theta}(1-\hat{\theta})}{n}}\right].$$

An approximate C.I. for θ of this kind with a confidence level of 95% is $[0, 0.8912]$. Since $\theta_0 = 0.9$ does not belong to the interval, the null hypothesis $H_0 : \theta \geq 0.9$ is rejected at a significance level of 5%.

A.5 Regression and Modeling

In statistics, wide use is made of *linear regression* and *model fitting*. The linear (regression) model is the simplest model trying to describe the (linear) linkage between one *response* variable Y and (one or) some *explicative* variables X_j, $j = 1, \ldots, p$, $p \geq 1$. We talk about *model fitting* when we are trying to investigate the probabilistic model beneath the r.v. X generating the data.

Let us consider again the algorithmic example introduced in Sect. A.1: an algorithm solving a set of instances of an optimization problem within a certain time limit. Let us assume now that the time limit is large enough that the algorithm always terminates with a solution found. Linear models can then be used to find a relationship between some parameters of the algorithm and its run time performance, while model fitting can be used to determine that the runtimes are, for example, exponentially distributed.

A.5.1 Linear Regression

The simplest model is the one linking two variables. The goal is to describe how Y varies on average as a function of X. To do that, we collect n independent observations of the bivariate variable $[X_i, Y_i]$, $i = 1, \ldots, n$ and fit a linear model that models the observed data "better." Once the linear model has been fitted, we can use it to determine whether there is a significant correlation between the response and the explicative variable(s), or try to predict the (average) value of $\mathrm{E}[Y]$ corresponding to a given a new observation x_0.

Thus, the model assumes that there is a linear relationship of the kind

$$Y_i = \beta_0 + \beta_1 x_i + \varepsilon_i,$$

where β_0 and β_1 are, respectively, the *intercept* and the *slope* of the line fitting the response data as a linear function of the explicative data, and ε_i is an r.v. (often known as *experimental error*) describing the variability of Y that does not depend on X. Note here that x_i is assumed to be the realization of the r.v. X_i. The errors are assumed to be an i.i.d. r.v. satisfying:

$$\mathrm{E}[\varepsilon_i] = 0, \qquad \mathrm{V}[\varepsilon_i] = \sigma^2, \qquad i = 1, \ldots, n.$$

These are the (minimal) assumptions on the ε_i's, i.e., we still have not specified the probability distribution of ε_i. An equivalent way to write the model is $\mathrm{E}[Y] = \beta_0 + \beta_1 X$, where the emphasis is on the fact that the fitting line models the expected value of Y as a function of X, rather than Y itself. In order to determine the model we require the estimates of β_0 and β_1. There are several ways to obtain these estimates, and here we will only refer to the *least squares error* estimation. Let $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$ be the prediction of $\mathrm{E}[Y]$ corresponding to the point x_i, $i = 1, \ldots, n$. The least squares estimates minimize the objective function

$$\mathrm{SSR}(\hat{\beta}_0, \hat{\beta}_1) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2.$$

In other words, $\hat{\beta}_0$ and $\hat{\beta}_1$ minimize the squared (Euclidean) distance among the observed and the predicted sets of points, so the fitted line "goes through" the observed data, which are points in $\mathbb{R}^2$. By differentiating $\mathrm{SSR}(\hat{\beta}_0, \hat{\beta}_1)$ with respect to $\hat{\beta}_0$ we obtain

$$\frac{\partial \mathrm{SSR}(\hat{\beta}_0, \hat{\beta}_1)}{\hat{\beta}_0} = -2 \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) = 0, \qquad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x},$$

where $\bar{x}$ and $\bar{y}$ are the sample means of, respectively, X and Y. By plugging in the estimate of β_0 and differentiating with respect to $\hat{\beta}_1$ we obtain

$$\frac{\partial \mathrm{SSR}(\hat{\beta}_1)}{\hat{\beta}_1} = -2 \sum_{i=1}^{n} [(y_i - \bar{y}) - \hat{\beta}_1 (x_i - \bar{x})](x_i - \bar{x}) = 0,$$

and therefore

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x})/n}{\sum_{i=1}^{n} (x_i - \bar{x})^2/n} = \frac{\mathrm{Cov}(X, Y)}{\mathrm{V}(X)},$$

where $\mathrm{V}(X)$ is the sample variance of X and $\mathrm{Cov}(X, Y)$ is the sample *covariance* between X and Y. The equation $\hat{y}_i = \bar{y} + \hat{\beta}_1 (x_i - \bar{x})$ gives the line fitting $E(Y)$ as a function of X.

The estimators of the parameters are unbiased:

$$\mathrm{E}[\hat{\beta}_1] = E\left[\frac{\sum_{i=1}^{n} (Y_i - \bar{Y})(x_i - \bar{x})}{\sum_{i=1}^{n} (x_i - \bar{x})^2}\right] = \frac{\sum_{i=1}^{n} (x_i - \bar{x}) \mathrm{E}[Y_i - \bar{Y}]}{\sum_{i=1}^{n} (x_i - \bar{x})^2}$$

$$= \frac{\sum_{i=1}^{n}(x_i - \bar{x})(\beta_0 + \beta_1 x_i - \beta_0 - \beta_1 \bar{x})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} = \beta_1;$$

$$\mathrm{E}[\hat{\beta}_0] = \mathrm{E}[\bar{Y} - \hat{\beta}_1 \bar{x}] = \mathrm{E}[\bar{Y}] - \mathrm{E}[\hat{\beta}_1]\bar{x} = \beta_0 + \beta_1 \bar{x} - \beta_1 \bar{x} = \beta_0.$$

Note that in the above equations we have stressed that X is not an r.v. (whereas Y is an r.v. since it depends on ε), and applied one property of the expected value for i.i.d. observation: $\mathrm{E}[\bar{Y}] = \mathrm{E}[\bar{Y}_i]/n = \beta_0 + \beta_1 \bar{x}$. It can be proved that the variance of $\hat{\beta}_1$ is

$$\mathrm{V}[\hat{\beta}_1] = \frac{\sigma^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}.$$

Accuracy of the Regression Model

Once the linear model has been fitted, we may ask something about the goodness of the fit. Clearly, a perfect fit should satisfy the condition $y_i = \hat{y}_i$ for all i; on the other hand, the worst model ever is such that the predicted values do not depend on X (hence there is no relationship between X and Y), e.g., $\hat{y}_i = k$ for all i, with k being a constant. The accuracy of the model is then evaluated by looking at the proportion of the variability of Y that is "explained" by X. The total sample *deviance* (the deviance of an r.v. is its variance multiplied by a constant) of Y can be decomposed as follows:

$$\mathrm{SST} = \sum_{i=1}^{n}(y_i - \bar{y})^2 = \sum_{i=1}^{n}(y_i \pm \hat{y}_i - \bar{y})^2$$
$$= \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2 + \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 - 2\sum_{i=1}^{n}(\hat{y}_i - \bar{y})(y_i - \bar{y}),$$

where the double product is equal to zero because it is equivalent to $\partial\mathrm{SSR}(\hat{\beta}_1)/\partial\hat{\beta}_1$. The total deviance can thus be written

$$\mathrm{SST} = \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2 + \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \mathrm{SSE} + \mathrm{SSR},$$

where SSE is the *explained* deviance (i.e., the variability of Y due to the model) and SSR is the residual deviance (i.e. the variability of Y not depending on X).

We have SST $=$ SSE when the observed points are already on a line (i.e., there is a perfect linear relationship between Y and X), and SST $=$ SSR when $\hat{\beta}_1 = 0$, so the fitted model is actually of the kind $\hat{y}_i = \bar{y}$. Thus, an index of the accuracy of the model is given by the ratio

$$R^2 = \rho(X,Y)^2 = \frac{\mathrm{SSE}}{\mathrm{SST}} = \frac{\hat{\beta}_1^2 \sum_{i=1}^{n}(x_i - \bar{x})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} = \frac{\mathrm{COV}(X,Y)^2}{\mathrm{V}[X]\mathrm{V}[Y]}.$$

Clearly, $0 \leq \rho(X,Y)^2 \leq 1$, and high values of $\rho(X,Y)^2$ indicate the presence of a strong *linear* relationship between X and Y. The index $\rho(X,Y)^2$ is the square of the *correlation coefficient*, which is a standardized measure of the covariance between X and Y and satisfies $-1 \leq \rho(X,Y) \leq 1$. The index R^2 is also known as the *coefficient of determination*.

Testing the Slope Coefficient

We have just seen that when $\hat{\beta}_1 = 0$ there is no (observed) linear relationship between Y and X, i.e., the variables are *uncorrelated*, but $\hat{\beta}_1$ is an estimate of the true parameter determining the (true) linear relationship β_1. Therefore it is important to evaluate whether β_1 is significantly different from 0 or not. That is, we want to perform a statistical test assessing the null hypothesis $H_0 : \beta_1 = 0$ against the alternative $H_1 : \beta_1 \neq 0$.

We require some further assumptions on the error distribution in order to perform a parametric test. Thus, the errors ε_i are assumed to be i.i.d. r.v.s with normal distributions. If $\varepsilon_i \sim N(0,\sigma^2)$, then also Y is normally distributed, specifically $Y_i \sim N(\beta_0 + \beta_1 x_i, \sigma^2)$. Being a linear combination of normal r.v.s, we have that $\hat{\beta}_1 \sim N(\beta_1, \sigma^2[\sum_{i=1}^n (x_i - \bar{x})^2]^{-1})$. If σ^2 is known, then the r.v.

$$T(\hat{\beta}_1, \beta_1) = \sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 \frac{\hat{\beta}_1 - \beta_1}{\sigma}} \quad \sim \quad N(0,1)$$

is a pivotal quantity, and therefore we can specify the acceptance/rejection regions of the test or obtain a two-sided p-value by considering the standard normal distribution as the null distribution of the test statistic. In practice, σ^2 is unknown and needs to be estimated from the data. A natural estimate of σ^2 is given by the (unbiased) variance estimator of the residuals

$$\hat{\sigma}^2 = \mathrm{V}(\hat{\varepsilon}) = \frac{1}{n-2} \sum_{i=1}^n \hat{\varepsilon}_i^2,$$

which is equal to the residual deviance divided by its degrees of freedom.[5]

Then if we replace σ^2 by its estimate, we have that the test statistic

$$T(\hat{\beta}, \beta_1) = \sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 \frac{\hat{\beta} - \beta_1}{\hat{\sigma}}} \quad \sim \quad t_2,$$

that is, $T(\hat{\beta}, \beta_1)$ has a Student t distribution with $n-2$ degrees of freedom (the Student t distribution with k degrees of freedom is defined as the ratio between

[5] It can be shown that the expected value of SSR is equal to $\sigma^2(n-2)$. It can also be proved that SSR $\sim \sigma^2 \chi^2_{n-2}$, i.e., SSR has a χ^2 distribution with $n-2$ degrees of freedom.

Z and $\sqrt{\chi_k^2/k}$, where $Z \sim N(0,1)$ and χ_k^2 is a chi-square r.v. with k degrees of freedom).

Thus we will reject the null hypothesis on β_1 for large values of $|T(\hat{\beta}, \beta_1)_{|\beta_1=0}|$.

Multiple Regression

The theory of simple linear regression can be easily extended to the more general case where $\mathrm{E}[Y]$ is modeled as a linear function of p explicative variables X_j, $j = 1, \dots, p$. That is, if we consider the linear model

$$Y_i = \beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} + \varepsilon_i.$$

With multiple linear regression we want to describe the variability of Y as a linear function of p explicative variables that may jointly influence the response. The assumptions on errors are as in the simple linear regression model, and it is possible to perform a statistical test on each slope coefficient β_j, $j = 1, \dots, p$. What changes here is that the estimates of the parameters are now functions of the *partial correlations* between Y and X_j, i.e., the correlation between Y and X_j computed after removing the correlations between the other explicative variables and Y. The decomposition of the total deviance still holds, but now the degrees of freedom of the residual deviance are $n-p-1$ (in simple linear regression $p = 1$). The adequacy of the model can be evaluated with an index of determination which accounts for the presence of p explicative variables. Indeed, if the number of variables increases the residual deviance decreases, even if none of the explicative variables is correlated with the response. This fact can be explained, for instance, by considering that the estimation of the parameters requires the solution of a system of $p+1$ equations. Therefore if $n = p+1$ there is only one solution that jointly satisfies all $p+1$ equations. Another intuitive example is given by the polynomial regression that has the form

$$Y_i = a + bx_i + cx_i^2 + dx_i^3 + \cdots + \varepsilon_i,$$

where the data (points of $\mathbb{R}^2$) are modeled by a polynomial function of X. It is known that there is only one line passing through two points, only one parabola passing through three points, etc. Thus, we must modify the simple coefficient of determination in order to take into account this geometric property. Define the *adjusted coefficient of determination*

$$R^2_{\text{adj}} = 1 - (1 - R^2)\frac{n-1}{n-p-1}.$$

An R^2_{adj} equal to 1 indicates perfect matching between the set of the responses and the set of predicted values (this happens when the nonadjusted R^2 is equal to 1). Note that, in some cases, R^2_{adj} could also be negative (e.g., when $R^2 = 0$): a similar result does not make sense in terms of the proportion of the variability explained by the model, and it must simply be interpreted as an index of a "terrible" model, i.e., a model where there is absolutely no correlation between the response and the explicative variables. The first thing to do when fitting a multiple regression model is to evaluate if at least one of the explicative variables has a significant correlation with the response. If this does not happens, the model is completely useless. We can express this situation with the null hypothesis $H_0 : \beta_1 = \beta_2 = \ldots, \beta_p = 0$ against the alternative $H_1 : \exists\, \beta_i \neq 0$. Note that, if the null hypothesis is true, then we are considering a model with intercept only (such a model always satisfies $\mathrm{SST} = \mathrm{SSR}$ and $R^2 = 0$). We can evaluate if our model is not significantly different from the intercept-only model by looking at their related explicative deviances (or residual deviances). Thus let SSR_0 and SSR_p be the residual deviances of respectively the null model and the model with p explicative variables that we are considering. From what we have said about the relationship between residual deviance and number of explicative variables considered, we can understand that $\mathrm{SSR}_0 \geq \mathrm{SSR}_p$. The difference $\mathrm{SSR}_0 - \mathrm{SSR}_p$ measures the increase of explained deviance that we obtain by adding p explicative variables to the null model. It can be shown that the r.v. $\mathrm{SSR}_0 - \mathrm{SSR}_p$ has a χ^2 distribution with p degrees of freedom. Moreover, it is independent of SSR_p and that the test statistic

$$F = \frac{(\mathrm{SSR}_0 - \mathrm{SSR}_p)/p}{\mathrm{SSR}_p/(n-p-1)} \quad \sim \quad F_{p\,;\,n-p-1},$$

has a Snedecor F distribution with p and $n - p - 1$ degrees of freedom. Small values of the test statistic are in agreement with the null hypothesis. The rejection region of the test has the form $[c(\alpha), +\infty)$, where $c(\alpha)$ is the $(1 - \alpha)$-quantile of the $F_{p\,;\,n-p-1}$ distribution. If the null hypothesis is not rejected, then none of the variables has a linear influence on the response.

The null hypothesis involving all parameters is not rejected, it is possible to test for the significance of each slope parameter β_j by applying the test statistic

$$T(\hat{\beta}_j, \beta_j) = \frac{\hat{\beta}_j - \beta_j}{\sqrt{\mathrm{V}(\hat{\beta}_j)}} \quad \sim \quad t_{n-p-1},$$

where $\mathrm{V}(\hat{\beta}_j)$ is the variance of the estimator of β_j, which in the general case is not easy to write in a closed form. As before, small values of $|T(\hat{\beta}_j, \beta_j)|_{\beta_j=0}|$ are not significant against the null hypothesis $\beta_j = 0$.

A.5.2 Model Fitting

There are some situations where one wants to know if the probabilistic model that has been assumed to generate the data is the correct one or not. For instance, one of the assumptions in linear regression is the normality of errors. It is usual to check whether the errors can be considered as normally distributed or not because, in the latter case, the inferential results (e.g., tests on coefficients) may not be completely reliable.

One descriptive method to check the normality of error is the *QQ plot*, which is a graphical representation of points whose coordinates are theoretical and empirical (observed) quantiles. By theoretical quantiles we mean the quantiles of the distribution that is assumed to hold for errors. Here the null hypothesis is $H_0 : \varepsilon_i \sim N(0, \sigma^2)$ against the alternative that the errors follow a nonspecified distribution F_ε. Clearly, the evaluation of the null hypothesis will be based on the empirical distribution function of the *residuals* $\hat{\varepsilon}_i = y_i - \hat{y}_i$, $i = 1, \dots, n$, which are realizations (or can be assumed as estimates) of the errors.

To obtain a QQ plot, order the residuals in increasing order; the jth ordered residual $\hat{\varepsilon}_{(j)}$ has an *empirical distribution function*

$$\hat{F}_n(\hat{\varepsilon}_{(j)}) = \frac{1}{n} \sum_{i=1}^{n} I(\hat{\varepsilon}_i \leq \hat{\varepsilon}_{(j)}),$$

and $\hat{F}_n(\hat{\varepsilon}_{(j)}) = j/n$ if there are no ties in the residuals (we assume that this happens with zero probability). Note that $\hat{F}_n(x)$ is defined for all x in $\mathbb{R}$ and that it is a step function. Then the theoretical quantiles, if errors are assumed to be normally distributed, are

$$z_{(j/n)} = \Phi^{-1}(j/n) \qquad j = 1, \dots, n.$$

The QQ plot represents the points whose coordinates are $[z_{(j/n)}, \hat{\varepsilon}_{(j)}]$. Since one of the properties of the normal distribution is that a linear combination of a normal r.v. is still normally distributed, if H_0 holds then the plotted points should lie along a line whose coefficients are approximately the coefficient of the linear combination linking the standard normal r.v. Z and the r.v. ε under testing. Thus, if one considers the standardized quantiles instead of the observed quantiles, what changes is just the equation of the theoretical line representing perfect agreement between the observed residuals and their theoretical quantiles.

The QQ plot is easy to interpret, but it lacks objectivity since the decision is made by visual inspection. There is a more scientific approach: the *Kolmogorov–Smirnov test*. hypothesis $H_0 : F_X(x) = F_0(x)$ against a nonspecified alternative $H_1 : F_X(x) \neq F_0(x)$, where $F_0(x)$ is a known distribution. The idea behind the test is that, if X is really distributed as $F_0(x)$, the theoretical and empirical distribution functions should be "close" to each other, and therefore the Kolmogorov–Smirnov test statistic

$$\text{KS} = \max_{x \in \mathbb{R}} |\hat{F}_n(x) - F_0(x)|$$

should be "small." Thus, this test has an acceptance region of the kind $[0, c(\alpha)]$, where $c(\alpha)$ is the $1 - \alpha$ quantile of the distribution of KS.

Recalling our starting example in Sect. A.1, we wish now to give a more precise indication of the runtime that the algorithm needs to solve an instance. To do this, we record the runtime of the algorithm on each specific instance (assuming it always finishes with a solution found). After the experiments we have a sample of run times, $y_1, y_2, \dots y_n$, one for each of n instances. We saw that the sample mean and the sample variance are some indicators of the distribution of runtimes; nevertheless, if we could find a theoretical distribution that fit the data well, the description would be more complete and, depending on the context, would allow us to exploit properties of the theoretical distribution. Moreover, for some distributions, such as heavy-tailed distributions, not all the moments are finite, implying that the sample mean and sample variance are erratic and not reliable descriptors. Hence, a more complete insight is definitely needed in these cases.

The typical procedure is the following: select a theoretical model, estimate its parameters, and then test the goodness of fit. Two models that we encountered in Section A.1.2, the exponential and the Weibull distribution, are used to describe life data are therefore also appealing to describe runtime distributions of algorithms. In particular, the Weibull distribution exhibits large flexibility due to the presence of three parameters in its model. Parameters in this kind of application are conveniently estimated by the *maximum likelihood method.*

The *likelihood* function $L(\cdot)$ is basically a density/probability function which is considered as a function of the parameter(s) rather than a function of the data. This is why $L(\cdot)$ is not a probability/density function. This choice is due to the fact that usually we try to choose a (parametric) probabilistic model once the data have been observed. Thus it is reasonable to choose the probabilistic model that, a posteriori, maximizes the probability of observing the data.

Suppose that we have the runtime results $y_1, y_2, \dots y_n$, and that we want to fit the distribution with an exponential model. The density of the exponential r.v. is

$$f_Y(y) = \lambda \exp\{-\lambda y\}, \qquad D_Y = (0, +\infty),\ \lambda > 0.$$

The density of a vector of i.i.d.[6] random variables $\mathbf{Y} = [Y_1, \dots, Y_n]$ is the product of their densities. Therefore

$$f_{\mathbf{Y}}(\mathbf{y}) = \prod_{i=1}^{n} f_{Y_i}(y_i) \qquad \mathbf{y} \in {D_Y}^n, f_{Y_i}(y_i) = f_Y(y_i)\ \forall\ i$$

where ${D_Y}^n$ is the n-dimensional Cartesian product of D_Y. The joint density of the vector in $y_1, y_2, \dots y_n$ (the observed data), viewed as a function of the parameter λ, is equal to the likelihood function

$$L(\mu, \lambda|\mathbf{y}) = \prod_{i=1}^{n} \lambda e^{-\lambda y_i} = \lambda^n \exp\left\{-\lambda \sum_{i=1}^{n} y_i\right\}.$$

According to the maximum likelihood method, the estimate of λ is the value that maximizes $L(\lambda)$. It is equivalent (and easier) to maximize the *log-likelihood*, i.e., the logarithmic transformation of $L(\lambda)$ (the logarithm function is monotone increasing)

[6] The requirement of identical distribution is not necessary in this definition. We have considered this case since the domain of $\mathbf{Y}$ is easier to describe and because it is part of the example.

$$\ell(\lambda|\mathbf{y}) = \log[L(\lambda|\mathbf{y})] = n\log(\lambda) - \lambda\sum_{i=1}^{n} y_i.$$

By differentiating $\ell(\lambda|\mathbf{y})$ with respect to λ:

$$\frac{\partial\ell(\lambda|\mathbf{y})}{\partial\lambda} = \frac{n}{\lambda} - \sum_{i=1}^{n} y_i = 0 \qquad \Rightarrow \qquad \hat{\lambda} = \frac{n}{\sum_{i=1}^{n} y_i}.$$

The likelihood method sometimes gives biased estimators (e.g., the estimator of the variance of a normal random variable). Therefore it is advisable to check whether the estimators of the parameters are biased or not (and, if so, modify the estimators ad hoc).

We know that the sum of n i.i.d. Gamma variables with parameters λ and $\nu = 1$ (recall that the exponential distribution is a Gamma with $\nu = 1$) has a Gamma distribution with parameters λ and n . Thus we may write the estimator as $\hat{\lambda} = n/W$, where $W \sim Ga(\lambda, n)$. Therefore

$$\begin{aligned}\mathrm{E}[\hat{\lambda}] = \mathrm{E}[nW^{-1}] &= \frac{n}{\Gamma(n)}\int_0^{+\infty} w^{-1}\lambda^n w^{n-1}\mathrm{e}^{-\lambda w}\mathrm{d}w \\ &= \frac{n\lambda}{\Gamma(n)}\int_0^{+\infty}\lambda^{n-1}w^{n-2}\mathrm{e}^{-\lambda w}\mathrm{d}w = n\lambda\frac{\Gamma(n-1)}{\Gamma(n)} = \lambda\frac{n}{n-1}.\end{aligned}$$

In this case the maximum likelihood estimator of λ is biased, therefore an unbiased estimator of λ is $\hat{\lambda} = (n-1)/\sum_i y_i$. Unfortunately, if one wants to apply the Kolmogorov–Smirnov test, the null distribution $F_0(x)$ must be completely specified. That is, we cannot estimate the parameter(s) of the distribution $F_0(x)$ from data, otherwise the KS test becomes conservative.

The KS test can also be applied in a two-sample problem: let $\mathbf{x}_1$ and $\mathbf{x}_2$ be two vectors of realizations of the r.v.s X_1 and X_2, respectively. The KS test can then be applied to assess the null hypothesis $X_1 \overset{d}{=} X_2$ against the alternative $X_1 \overset{d}{\neq} X_2$.

In this case the test statistic is equal to $\mathrm{KS} = \max_{x\in\mathbb{R}} |\hat{F}_{n_1}(x) - \hat{F}_{n_2}(x)|$, where $\hat{F}_{n_j}(x)$ is the empirical cdf of the jth sample at point x, $j = 1, 2$. Figure A.7 shows an example of the KS statistic when $\mathbf{x}_1$ are ten realizations from $X_1 \sim \mathrm{Exp}(1)$ and $\mathbf{x}_2$ are 20 realizations from $X_2 \sim \mathrm{Exp}(2)$. In this case the value of the test statistic is $\mathrm{KS} = 0.35$ and the related p-value is equal to 0.3686, so we do not reject the null hypothesis.

References

Davison AC (2008) Statistical Models. Cambridge University Press, New York

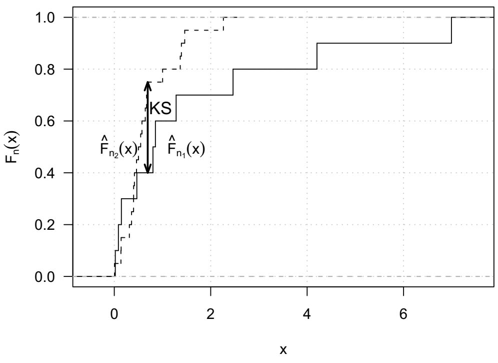


Fig. A.7: The Kolmogorov–Smirnov test statistic for two distributions of data $X_1 \sim \text{Exp}(1)$ and $X_2 \sim \text{Exp}(2)$

Johnson NL, Kotz S (1970) Distributions in statistics. Wiley Series in Probability and Mathematical Statistics, New York, NY

Mood A, Graybill F, Boes D (1974) Introduction to the Theory of Statistics. McGraw-Hill, Auckland

Index