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