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

T. Bartz-Beielstein et al. (eds.), *Experimental Methods for the Analysis of Optimization Algorithms*, DOI 10.1007/978-3-642-02538-9_2,

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

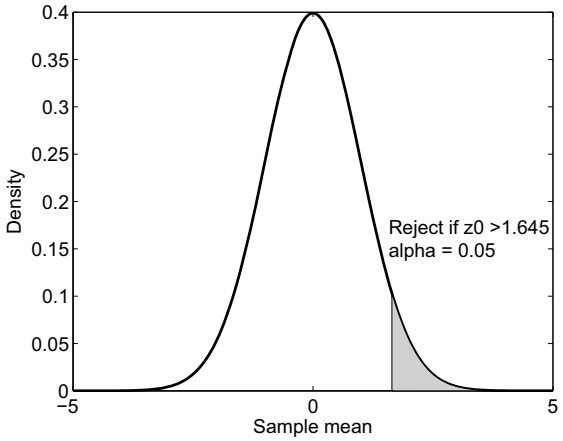

Fig. 2.1: Hypothesis testing. Consider $n = 100$, $\sigma = 1$, $\mu_0 = 0$, and $\alpha = 0.05$. The critical region is $[1.645, \infty[$, and $z_0 = \overline{x} \times 10$, cf. (2.1). For instance, the testing rule leads to a rejection of the null hypothesis if the difference of observed means is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

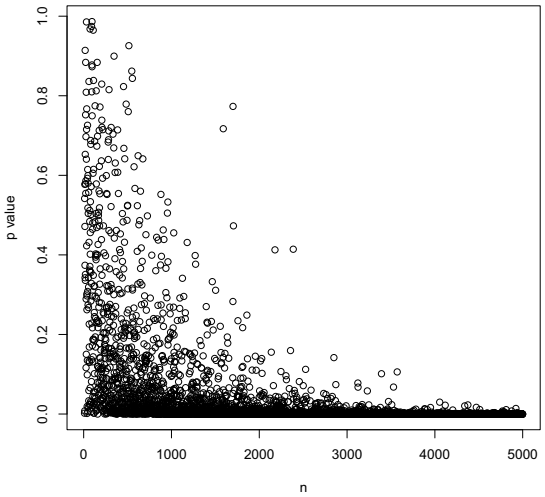

Fig. 2.2: Increasing the number of experiments can reduce the p-value. This figure illustrates the situation described in Example 2.3. The number of repeats is varied from $n = 10$ to 5000. For each setting, n realizations of the random variables Y_A and Y_B are generated and the corresponding p-value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

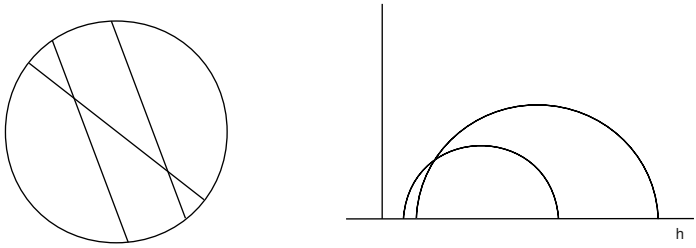

Fig. 2.3: Two models for hyperbolic geometry. *Left:* The Klein model. *Right:* The Poincaré half-plane model. Hyperbolic lines are chords of the circle in the Klein model; they are rays or half-circles in the Poincaré half-plane model

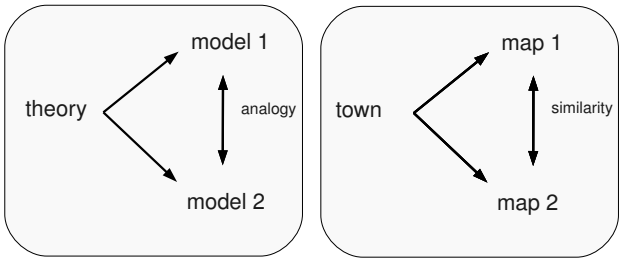

Fig. 2.4: *Left:* Analogy interpreted as isomorphism. *Right:* Similarity as a concept for real-world (representational) models. Maps, which represent the same geographical area, are similar, not necessarily isomorphic. Consider, e.g., a map which illustrates railway lines and a map which illustrates 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($\mathbf{x}$, σ);
$y_p := f(\mathbf{x_p})$;
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 σ 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

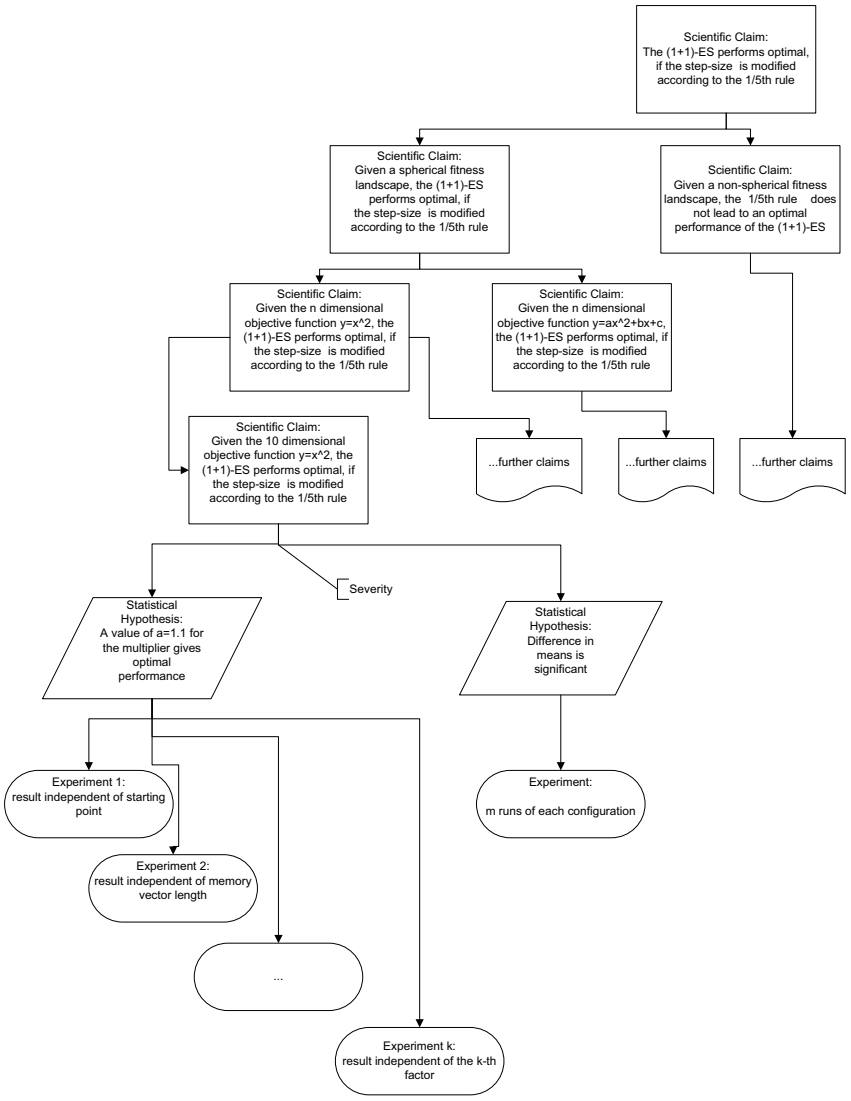

Fig. 2.5: Primary scientific hypothesis broken into local hypotheses. Rectangles denote elements from the primary model, elements from the experimental model use parallelograms, and ellipses are used to represent elements from the data model

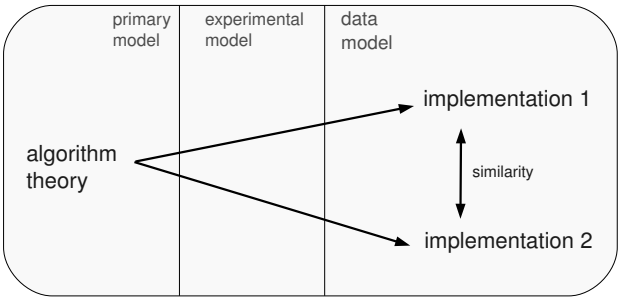

Fig. 2.6: Refining the theory–model relation from Fig. 2.4 by introducing a model hierarchy

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.

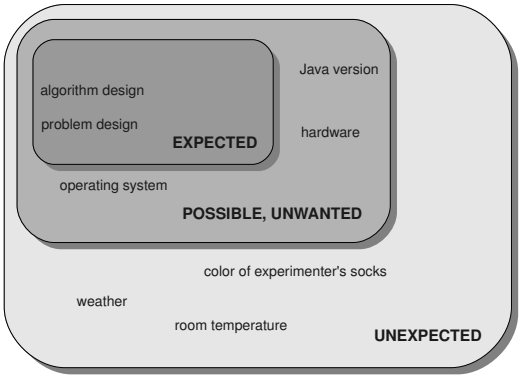

Fig. 2.7: Classification of factors in the experimental approach. a) Factors which belong to the algorithm and problem design should have an effect on the experimental outcome. b) Sometimes, side-effects or unwanted effects occur during experimentation. The experimenter should eliminate the influence 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-2: Statistical hypotheses
SPO-3: Experiments
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 ex-

perimental 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