

Theory of Iterative Optimization Heuristics: From Black-Box Complexity over Algorithm Design to Parameter Control

Carola Doerr

▶ **To cite this version:**

Carola Doerr. Theory of Iterative Optimization Heuristics: From Black-Box Complexity over Algorithm Design to Parameter Control. Neural and Evolutionary Computing [cs.NE]. Sorbonne Université, 2020. tel-03168778

HAL Id: tel-03168778
https://hal.sorbonne-universite.fr/tel-03168778

Submitted on 26 Mar 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

SORBONNE UNIVERSITÉ

Theory of Iterative Optimization Heuristics: From Black-Box Complexity over Algorithm Design to Parameter Control

by

Carola DOERR

for the diploma

Habilitation à Diriger des Recherches (HDR)

Committee Members

Laetitia Jourdan	rapporteur	Université de Lille, France
Jonathan E. Rowe	rapporteur	The University of Birmingham, UK
Carsten Witt	rapporteur	Technical University of Denmark, Denmark
Carlos Artemio Coello Coello	jury member	CINVESTAV-IPN, Mexico
Christoph Dürr	jury member	CNRS and Sorbonne Université, France
Günter Rudolph	jury member	TU Dortmund University, Germany
Marc Schoenauer	jury member	INRIA, France
Lothar Thiele	jury member	ETH Zürich, Switzerland

Date of the Defense: December 18, 2020
President of the Jury: Christoph Dürr

Abstracts in English and French

Abstract (English Version)

Sampling-based optimization heuristics are algorithms which aim to find good solutions for problems that cannot be solved through exact solution strategies. They are particularly useful for the optimization of problems that are not given as explicit formulae, but which require computer simulations or physical experiments to assess the quality of a proposed solution. In such settings, sampling-based optimization heuristics are essentially the only means to obtain good solutions. Another important application of sampling-based optimization heuristics is in the optimization of problems that are too complex to be solved analytically, and which remain intractable for problem-specific algorithm designs.

An important sub-class of sampling-based optimization heuristics are iterative optimization heuristics (IOHs). IOHs aim at finding good solutions by a sequential process of evaluating solution candidates, using the obtained information to adjust the strategy by which the next samples are generated, and iterating this process until a termination criterion is met.

With my research, I aim to contribute to our understanding of the working principles that drive the performance of IOHs. More precisely, I aim at understanding which strategies work well for which types of problems. To this end, I study the efficiency of existing IOHs, and I compare these performances to that of a best possible solver for the given problem. For the latter, I develop and analyze complexity models that allow us to quantify the influence of certain algorithmic choices, such as the size of memory used, the strategy by which solutions can be sampled, the type of information used to select the next sampling strategies, etc.

This thesis summarizes some of my research results obtained in the last nine years, since the completion of my PhD studies. We first derive improved (and often tight) bounds for the black-box complexity of the two best known benchmark problems in the theory of IOHs, OneMax and LeadingOnes, and this for various black-box models. We then demonstrate how insights obtained from such black-box complexity studies can inspire the design of efficient optimization techniques. An important result obtained through this approach is a rigorous example for a situation in which a non-static choice of the control parameters of an IOH yields a super-constant performance gain over any possible static parameter setting. This result has revived theoretical research on parameter control, which has seen significant advances since. We discuss some examples in this thesis. We conclude by discussing the role of algorithm benchmarking as a bridge between theoretical and empirical research of IOHs.

Résumé (French Version of the Abstract)

Les heuristiques d'optimisation fondées sur l'échantillonnage sont des algorithmes qui visent à trouver de bonnes solutions aux problèmes ne pouvant pas être résolus par des stratégies de solution exactes. Elles sont particulièrement utiles pour l'optimisation de problèmes qui ne sont pas décrites par des formules closes explicites, mais qui nécessitent des simulations informatiques ou des expériences physiques pour évaluer la qualité d'une solution proposée. Dans de tels contextes, les heuristiques d'optimisation fondées sur l'échantillonnage sont essentiellement le seul moyen d'obtenir de bonnes solutions. Une autre application importante des heuristiques d'optimisation fondées sur l'échantillonnage est l'optimisation des problèmes qui sont trop complexes pour être résolus de manière analytique et qui qui n'admettent pas des des algorithmes spécifiques.

Une sous-classe importante d'heuristiques d'optimisation fondées sur l'échantillonnage sont les heuristiques d'optimisation itérative (I.O.H.). Les IOH cherchent à trouver de bonnes solutions par un processus séquentiel d'évaluation de solutions candidates, en utilisant les informations obtenues pour ajuster la stratégie par laquelle les candidates suivants sont produites, et en itérant ce processus jusqu'à ce qu'un critère de terminaison soit satisfait.

Avec mes recherches, je mefforce de contribuer à notre compréhension des principes opérationnels qui déterminent la performance des IOH. Plus précisément, je cherche à comprendre quelles stratégies fonctionnent bien pour quels types de problèmes. Pour cela, j'étudie l'efficacité des IOH existants, et je compare ces performances à celle d'un meilleur solveur possible pour le problème donné. Pour ces derniers, je développe et analyse des modèles de complexité qui permettent de quantifier l'influence de certains choix algorithmiques, comme la taille de la mémoire utilisée, la stratégie par laquelle les solutions peuvent être échantillonnées, le type d'informations utilisées pour sélectionner les prochaines stratégies d'échantillonnage, etc.

Ce manuscrit résume certains de mes résultats de recherche obtenus au cours des neuf dernières années, depuis la fin de mes études de doctorat. Nous dérivons d'abord des bornes améliorées (et souvent serrées) pour la complexité en boîte noire des deux problèmes de référence les plus connus dans la théorie des IOH : OneMax et LeadingOnes, et ce pour divers modèles en boîte noire. Nous démontrons ensuite comment les informations obtenues à partir de ces études de complexité en boîte noire peuvent inspirer la conception de techniques d'optimisation efficaces. Un résultat important obtenu grâce à cette approche est un exemple rigoureux de situation dans laquelle un choix non statique des paramètres de contrôle d'une IOH produit un gain de performance super-constant par rapport à tous les réglages de paramètre statique possibles. Ce résultat a relancé la recherche théorique sur le contrôle des paramètres, qui a connu des avancées significatives depuis. Nous discutons quelques exemples dans ce manuscrit. Nous concluons en discutant le rôle du benchmarking d'algorithmes comme pont entre la recherche théorique et empirique des IOH.

Contents

Abstracts in English and French **iii**

1 Introduction **1**

1.1 Iterative Optimization Heuristics (IOHs) 2

1.2 The Role of Theoretical Research for the Analysis of IOHs 4

1.3 Evolutionary Algorithms – Dictionary and Examples 5

1.4 Benchmark Algorithms and Problems Frequently Appearing in this Thesis 9

1.5 Performance Measures 10

1.6 Outline of the Thesis 12

2 Black-Box Complexity Theory Continued **13**

2.1 Black-Box Complexity – Definition 13

2.2 Black-Box Complexity – Motivation 14

2.3 Results for the Unrestricted Model 15

2.4 Results for Restricted Black-Box Models 18

2.5 Outlook 27

3 From Black-Box Complexity Theory to Algorithm Design **29**

3.1 The $(1 + (\lambda, \lambda))$ GA– An $o(n \log n)$ Genetic Algorithm for OneMax 30

3.2 Randomized Local Search with Self-Adjusting Mutation Strengths 32

4 From Algorithm Design to Parameter Control **35**

4.1 Self-Adjusting Parameter Values for the $(1 + (\lambda, \lambda))$ GA 36

4.2 Classification of Parameter Control Methods 37

4.3 One-Fifth Success Rule Revisited: Self-Adjusting Mutation Rates for the (1+1) EA . . 38

4.4 Parameter Control for Multi-Valued Decision Variables 41

5 Selected Other Topics **43**

5.1 A Simple Proof for the Usefulness of Crossover 43

5.2 Re-Optimization for Structurally Similar Problem Instances 44

5.3 One-Shot Optimization for Continuous Optimization 44

5.4 IOHprofiler - A Versatile Benchmarking Environment 46

6 Outlook **49**

Curriculum vitæ **51**

Bibliography **57**

Chapter 1

Introduction

Many, if not most, real-world optimization problems are not given as explicit mathematical formulae, but require (possibly expensive) evaluations to assess and to compare the quality of different alternatives. Classical examples for such problems are settings in which the evaluation of a suggested solution requires a computer simulation or a physical experiment, such as a crash test or a clinical study. Computer Science studies the solution of such in-explicitly given problems under the notion of **black-box optimization**. Black-box optimization problems are ubiquitous. They appear in almost any scientific discipline and across almost all industrial sectors, with applications ranging from product design and engineering, over maintenance, medial treatment scheduling, agriculture, systems biology, and network optimization to numerous other use cases. In recent years, black-box optimization has gained visibility in the broader Computer Science context, because of its importance in Machine Learning and, more generally, in Artificial Intelligence, where many of the core problems such as the design of (deep) neural networks, (hyper-)parameter optimization, etc. are essentially black-box optimization problems.

Black-box optimization problems can only be solved by *sampling-based heuristics*, i.e., algorithms which (deterministically or randomly) generate solution candidates, evaluate them, and use the so-obtained knowledge to recommend one or more alternatives. Depending on the adaptive behavior of the algorithms, we distinguish two main classes of sampling-based heuristics:

- **Iterative Optimization Heuristics (IOHs):** These algorithms proceed in rounds. After each round, the heuristic updates the strategy by which the next solution candidates are generated. See Section 1.1 for a more detailed description and examples.
- **One-Shot Optimization Algorithms:** In some situations, an adaptive procedure as required by IOHs is not possible, e.g., because of prohibitive evaluation times or cost. In this case, one-shot optimization techniques need to be chosen. These non-adaptive algorithms select the set of solution candidates prior to the first evaluation. Random sampling, Latin Hypercube designs [MBC79], and quasi-random point sets such as low-discrepancy point sets [Nie92, DP10] are classical examples for such one-shot optimization strategies.

The main focus of this thesis is on *iterative* optimization. A few results for one-shot optimization, however, will be summarized in Sections 5.3.

Before reducing our scope to IOHs, we briefly mention that the application of sampling-based optimization algorithms is not restricted to black-box optimization alone. Indeed, sampling-based optimization heuristics have proven useful in a number of contexts in which the problem to be solved *does* have an

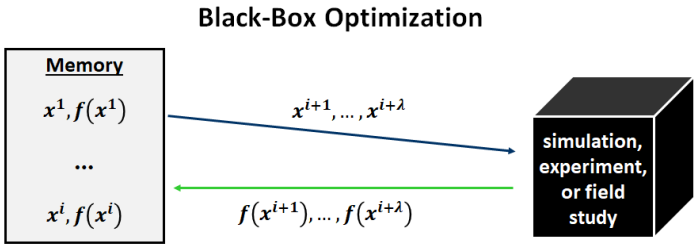


Figure 1.1: Black-box optimization problems require sampling-based algorithms. These algorithms treat the problem instance as an oracle, which reveals (absolute or relative) information about the quality of the solution candidates. Depending on the use or not of adaptive sampling strategies, we distinguish between *iterative optimization heuristics (IOHs)* and *one-shot optimization algorithms.* The main focus of this thesis is on IOHs.

explicit formulation (*white-box setting*) or in which *some* information about the interaction between the decision variables is known (*gray-box setting*). In such applications, it typically comes handy that sampling-based algorithms can easily be applied to new problems, without requiring deep knowledge about the problem at hand, so that off-the-shelf implementations can be used to get first meaningful results. Certainly, there are also cases in which an explicitly given problem simply remains intractable for rigorous algorithmic solutions, possibly despite significant research efforts. The low autocorrelation binary sequence problem [Gol72, PM16] is a well-known example of the latter type.

1.1 Iterative Optimization Heuristics (IOHs)

As sketched above, IOHs are adaptive sampling strategies, which adjust their behavior during the optimization process. That is, IOHs are initialized by selecting a set of candidate solutions that are evaluated in a first iteration. Once the quality of these *search points* is known, a second set of solution candidates is selected, then evaluated, and so on, until some stopping criterion is met; for example, when a solution of sufficient quality has been found, when a time budget is exhausted, when no progress has been observed for some time, or when the diversity of the solutions has fallen below some threshold. With each iteration, the heuristic collects more information about the problem instance at hand, which it can use to focus the search on the most promising regions in the decision space.

Algorithm 1 summarizes the structure of an IOH. Concrete examples will be presented in Sections 1.3 and 1.4. IOHs can, but do not need to be randomized. When instantiating all distributions in the framework of Algorithm 1 by degenerated ones (one-point distributions), the algorithm is deterministic. The focus of our work, however, is almost exclusively on *randomized IOHs.*

IOHs can be loosely categorized into the following classes:[1]

- **Single-point IOHs:** This class subsumes algorithms which essentially keep one point as the center of search and evolve both the mean and the shape of the sampling distributions. We distinguish two main sub-classes:

[1]Note here that this classification is rather informal, as the boundaries between the different classes are fuzzy, and hybridization and/or sequential execution ("chaining") of one or several approaches is not uncommon. Readers interested in more detailed summaries of these algorithms can find many books and tutorials on this matter. Recommended surveys with numerous pointers to relevant literature are [SEBB18, BLS13, BPS03].

Algorithm 1 Iterative optimization heuristics (IOHs) alternate between the generation of new solution candidates and their evaluation. Information obtained from these evaluations is used to adjust the strategy by which the next candidates are generated. Many, but not all, IOHs are randomized.

1: $t \leftarrow 0$ ▷ iteration counter
2: $\mathcal{H}(0) \leftarrow \emptyset$ ▷ search history information
3: choose a distribution $\Lambda(0)$ on $\mathbb{N}$ ▷ distribution of the number of samples
4: **while** termination criterion not met **do**
5: $t \leftarrow t + 1$
6: sample $\lambda(t) \sim \Lambda(t-1)$ ▷ nbr. of points to be evaluated in this iteration
7: Based on $\mathcal{H}(t-1)$ choose a distribution $D(t)$ on $\mathcal{S}^{\lambda(t)}$ ▷ choice of sampling distribution
8: sample $\left(x^{(t,1)}, \ldots, x^{(t,\lambda(t))}\right) \sim D(t)$ ▷ candidate generation
9: evaluate $f\left(x^{(t,1)}\right), \ldots, f\left(x^{(t,\lambda(t))}\right)$ ▷ function evaluation
10: choose $\mathcal{H}(t)$ and $\Lambda(t)$ ▷ information update

  - **Local search algorithms:** Local search algorithms move from one solution candidate to the next by comparatively small perturbations. Where a neighborhood structure exists, such as the Hamming neighborhoods in the search space $\{0,1\}^n$, the search moves along the edges in this neighborhood structure, i.e., any two consecutively evaluated points are neighbors. We further distinguish *greedy* heuristics (such as first improvement or best improvement strategies) from *non-greedy* search heuristics, such as Simulated Annealing [KGV83], Threshold Accepting [DS90], or Tabu Search [Glo86]. Whereas greedy heuristics always continue the search in a best-so-far solution, the non-greedy ones may continue the search in a worse than best-so-far solution. The probability to "accept" an inferior center of search decreases with the difference in quality (and for a given discrepancy, it typically also decreases with time). Non-greedy search heuristics therefore aim to avoid the risk of getting stuck in a local optimum by temporarily accepting worse solutions, in the hope to traverse the "valleys" in the fitness landscape.
  - **Global search algorithms:** Another strategy to overcome local optima is non-local sampling. That is, the probability distribution from which *global search heuristics* sample the solution candidates assigns positive values also to solutions that are not direct neighbors of the current center of search. The probability to sample a point at a certain radius may change over time, so that the heuristic can converge from an *exploratory* initial phase to an *exploitation* stage, in which the search behaves more locally. Single-point (also known as *single-trajectory*) evolutionary algorithms such as the $(1+\lambda)$ EAs (which we will introduce in Section 1.4) or Variable Neighborhood Search [MH97] are classical examples for global search algorithms.

- **Population-based algorithms:** Algorithms maintaining a set of solution candidates to decide center and shape of the next sampling distributions are referred to as population-based heuristics. Most evolutionary algorithms and genetic algorithms fall into this category [Bäc96, ES03]. Other classes of population-based heuristics are estimation of distribution algorithms (EDAs [MP96, LL02]), the covariance matrix adaptation evolution strategies (CMA-ES [HO01]), differential evolution (DE [SP97]), and Swarm intelligence algorithms such as Particle Swarm Optimization [KE95] and Ant Colony Optimization [Dor92].

- **Surrogate-based algorithms:** Surrogate-based algorithms use the evaluated search points to approximate the true optimization problem f (or parts of it) by a surrogate $\hat{f}$, with the idea that by optimizing $\hat{f}$ the algorithm can save calls to f. That is, the model $\hat{f}$ is used to propose the point(s) to be evaluated next. The model $\hat{f}$ is frequently updated, e.g., after each iteration. One of the best known surrogate-based heuristics is the efficient global optimization algorithm (EGO [JSW98]), also referred to as Bayesian optimization. Hybridization of other

Theoretical Analysis	**Empirical Evaluation**
+ performance *guarantees*	numerical observations
+ proof tells you the *reason*	only observe numbers
+ bounds for whole *classes of problems*	individual instances only
+ *implementation-independent*	results can depend on implementation
time-consuming and arbitrarily difficult	+ *fast*, often *easy* to implement
simple benchmark problems	+ *complex problems*
simple heuristics/control schemes	+ *complex heuristics/control schemes*
limited precision	+ *exact numbers*
main scope: asymptotic behavior	main scope: concrete range of dimensions

Table 1.1: Comparison of theoretical and empirical research in black-box optimization, adapted from our GECCO tutorial *Theory for Non-Theoreticians* [DD16b]

IOHs with local surrogate models to select which points to further evaluate are referred to as *surrogate-assisted* IOHs.

1.2 The Role of Theoretical Research for the Analysis of IOHs

A plethora of IOHs exist, and their number is ever growing – to a point where we can hardly reconstruct whether a certain algorithm has previously been used or not, see discussions in [CSD20, Sör15] and references mentioned therein. Most of these heuristics are developed with a particular application in mind, and one of the key interests of our research domain is to understand to what extent the algorithm (or an algorithmic idea) can be useful beyond the particular application that it has been designed for. Put differently, **one, if not *the* most important objective in the analysis of IOHs is to analyze which approaches work well on which type of problems, and *why*.** Two main approaches to tackle this question exist:

Empirical Approaches/Algorithm Benchmarking: The empirical analysis and comparison of sampling-based optimization heuristics is often referred to as *benchmarking*. In contrast to a purely performance-oriented, competitive development of efficient solvers, benchmarking aims at understanding *why* certain algorithms behaves well (or not) on a given (set of) problem(s) or problem instance(s). Designing sound benchmark studies is a tedious task and almost every step required in a benchmarking study is subject to active research. This includes the selection of the benchmark problems and the problem instances, of the algorithms to be executed (and their configuration), of the performance measures and the statistics used to analyze the results, of the data to be logged during execution (i.e., *which* data shall be tracked?), of the granularity of this data (i.e., *how often* do we record the data?), and of an appropriate visualization of this data.

Mathematical Approaches/Theory of Sampling-Based Heuristics: An alternative way towards understanding the behavior of IOHs is their analysis via *mathematical means*. Often subsumed under the umbrella term *theory of evolutionary computation,* this sub-domain aims at proving performance guarantees or statements about the search process of IOHs with mathematical rigor. In this thesis, we use the terms "theory", "theoretical guarantee", etc. with such a mathematical approach in mind. In the broader literature, however, different interpretations of these expressions are in use. Notably, some authors subsume under the term "theory" all research not directly involving industrial or academic applications.

We should note here that not only the approaches taken to analyze IOHs are different, but also the *scope*. Where, in principle, empirical research is mostly bounded by the availability of human and computational resources, theoretical research of IOHs quickly faces methodological challenges. A key

difficulty in analyzing randomized IOHs are the complex dependencies of the random variables that are needed to describe their optimization behavior and their performance traces. Theory of IOHs is therefore often restricted to rather simple problems and algorithms, which do not necessarily represent very well the complexity of the problems and algorithms solved and applied in practice. Of course, our hope is that using the theoretical approach we can nevertheless gain insight into the working principles of IOHs that can be generalized to more complex algorithms and problems. More broadly, we hope that the "theory way of thinking" can derive insights that are useful for practical purposes. This situation is comparable to the relationship between other sub-domains of theoretical and applied Computer Science.

Despite all differences in the type and scope of results that can be obtained through either approach, empirical and mathematical research should not be seen as competing approaches. They rather *complement* each other, by offering different views on the performance and the search behavior of IOHs. Table 1.1 summarizes the advantages of theoretical and empirical approaches, respectively. This comparison clearly shows that we cannot rely on either one approach only, if we aim to get a rather complete picture of the strengths and weaknesses of different IOHs.

Reflecting my main research activities in the last nine years, this thesis focuses almost exclusively on mathematical approaches. A few empirical results will nevertheless be mentioned, e.g., where they have crucially inspired our analyses or where they could demonstrate the applicability of our results beyond the setting covered by the theoretical analyses. Some further empirical results, which are not necessarily related to theoretical results, will be mentioned in Sections 5.3.2 and 5.4. In particular, we will present in Section 5.4 our modular benchmarking platform, IOHprofiler, which we have built with the goal to facilitate both empirical and theoretical research on IOHs, and the communication between the two sub-domains.

By design, this thesis is strongly biased towards my own contributions to the theory of sampling-based optimization heuristics. Readers interested in a broader overview of the field are referred to the books [DN20, AD11], where summaries of state-of-the-art results and methods for the analysis of IOHs can be found. The book [NW10] surveys tools and results for combinatorial optimization, and the book [Jan13] is recommended for a gentle introduction to running time analysis of evolutionary algorithms.

1.3 Evolutionary Algorithms – Dictionary and Examples

In the broader Computer Science context, research on IOHs is often subsumed under the term "evolutionary computation", "bio-inspired computing", or "nature-inspired optimization". Classically, these terms used to refer to algorithms that were inspired – one way or the other – by phenomena observed in nature. Today, conferences on evolutionary computation cover the whole spectrum of sampling-based optimization techniques, and – where applicable – also the hybridization with exact solvers. It is therefore important to keep in mind that "evolutionary computation" does not necessarily require nor aim for nature-inspired design principles.

In the context of this thesis, biological or other nature-driven inspiration has no relevance – with one important exception, and this is the terminology used in both this thesis and, more broadly, in the publications of our research domain, which goes back to the times in which the biological inspiration was still a major "selling point" of a new algorithmic idea. While a general trend towards avoiding such community-specific terminology can be observed, several terms are still very actively used. This leads to a rather unfortunate situation in which algorithms and results are not always immediately accessible to non-experts. Despite best efforts, the works presented in this thesis are no exception to this rule, as we always need to balance between addressing a general audience and the possibility for the experts to quickly identify the relationship to previous works. We therefore briefly summarize and "translate" the most commonly used community-specific terms used in this thesis:

- The *fitness function* or *objective function* refers to the problem instance at hand. Even if not modeled explicitly, there exists a function $f : \mathcal{S} \to \mathbb{R}^d$, which maps the admissible decision alternatives $x \in \mathcal{S}$ to the values that are obtained through the black-box evaluations. The value $f(x)$ of a search point $x \in \mathcal{S}$ is also referred to as its *fitness*, its *function value*, or its *objective value*.
- An *individual* is an element of the decision space, i.e., a point $x \in \mathcal{S}$. To express a temporal component, individuals are sometimes further specified into *parents* and *offspring*. In this thesis, we typically speak of *search points* or *solution candidates* when referring to the elements of the decision space, and we also call $\mathcal{S}$ the *search space*. A *population* is a set of search points. To indicate the temporal component, we distinguish the *parent population* from the *offspring population*. That is, the parent population is the set or a subset of the points that have been evaluated before the current iteration, whereas the offspring population is the one that is generated (and in most cases also evaluated) in the current iteration.
- A *mutation operator* is a family $(D(\cdot \mid x))_{x \in \mathcal{S}}$ of unary probability distributions over the search space $\mathcal{S}$. Given a point $x \in \mathcal{S}$, the mutation operator generates a new search point by sampling from $D(\cdot \mid x)$. The probably most commonly known mutation operator is the one that samples a random neighbor (we will call this operator $\mathrm{flip}_1(\cdot)$ in this thesis, see discussion below).
- A *crossover operator* or *recombination operator* is a family $(D(\cdot \mid x^1, \ldots, x^k))_{(x^1,\ldots,x^k) \in \mathcal{S}^k}$ of k-ary probability distributions over the search space $\mathcal{S}$ with $k > 1$. A well-known example is the coordinate-wise majority vote. More examples will be provided below.

Notation and assumptions: We use below and in the remainder of this thesis the following notation. By $[a..b]$ we denote the set of integers k satisfying $a \leq k \leq b$. Unless stated otherwise, we assume that the optimization problem is a problem of the form $f : \{0,1\}^n \to \mathbb{R}$, i.e., a so-called *pseudo-Boolean function*. In particular, we assume the search space to be the full n-dimensional hypercube $\{0,1\}^n$ (when considering constrained problems, we typically use penalization approaches, but we focus in this thesis on *unconstrained* optimization problems, i.e., we assume all points $x \in \{0,1\}^n$ to be feasible). We restrict our attention to *single-objective* optimization, and, unless stated otherwise, we assume *maximization* as objective. For $x \in \{0,1\}^n$ we write $x = (x_1, \ldots, x_n)$. By *flipping* a bit x_i we mean to replace it by $1 - x_i$. For two strings x and y we denote by $z = x \oplus y$ the bitwise XOR, i.e., $z_i = 0$ if $x_i = y_i = 0$ or $x_i = y_i = 1$, and $z_i = 1$ otherwise. For an integer $n \in \mathbb{N}$ we denote by S_n the symmetric group of the set $[1..n]$, i.e., the set of all permutations (one-to-one maps) $\sigma : [1..n] \to [1..n]$.

Example: A Family of $(\mu + \lambda)$ Genetic Algorithms (GAs). Since it will be useful for later discussions, we present in Algorithm 2 a framework for a class of so-called $(\mu + \lambda)$ genetic algorithms (GAs). These algorithms are initialized by sampling and evaluating μ search points, which form the initial population. In each iteration, λ new points are evaluated. Each of these λ "offspring" is generated by first deciding whether or not to do a crossover operation. In the version presented in Algorithm 2, crossover happens with probability p_c (line 7). If crossover is chosen, two points are selected from the current population (line 8), and a new point is generated by applying a crossover operator to them. Subsequently, a random decision is taken to decide whether or not the so-created point undergoes a mutation step (line 11). If crossover was not chosen in line 7, the new candidate solution is generated by first selecting a point from the current population P (line 13) and then applying a mutation operator to it. The new search point is then evaluated (line 15) and inserted into the offspring population O. When all λ offspring have been evaluated, the algorithm updates its control parameters and operator choices and selects which of the $\mu + \lambda$ points from the set $P \cup O$ to keep for the next iteration. Note that in all the above and below, the sets P and O are, strictly speaking, *multi-sets*, i.e., we allow points to appear more than once. When we count the number of elements in the set, each point contributes with the number of times it appears in the multi-set. Likewise, when we say to remove a point, we remove only one of the possibly multiple copies.

Algorithm 2 A Family of $(\mu + \lambda)$ Genetic Algorithms (GAs)

```
1:  P ← InitialSampling(μ)                                          ▷ initialization
2:  evaluate the μ points in P
3:  while termination criterion not met do
4:      O ← ∅
5:      for i = 1, ..., λ do
6:          Sample r_c ∈ [0, 1] u.a.r.                              ▷ proba. of crossover is p_c
7:          if r_c ≤ p_c then                                       ▷ crossover step
8:              (x, y) ← SelectC(P, 2)                              ▷ parent selection
9:              z ← Crossover(x, y)                                 ▷ (pre-)candidate generation
10:             Sample r_m ∈ [0, 1] u.a.r.                          ▷ random decision: mutation after crossover?
11:             if r_m ≤ p_m then z ← Mutation(z)
12:         else                                                    ▷ mutation step
13:             x ← SelectM(P, 1)                                   ▷ parent selection
14:             z ← Mutation(x)                                     ▷ candidate generation
15:         Evaluate z
16:         O ← O ∪ {z}                                             ▷ z added to offspring population
17:     Update the control parameters and operator choices
18:     P ← Replace(P, O, μ)          ▷ decide which μ points to keep for the next iteration
```

We did not specify above the operators that the $(\mu + \lambda)$ GA makes use of. We briefly summarize a few standard choices below. Note, though, that – despite the great number of combinations it already offers – this list is very far from being exhaustive. Notably, our selection is clearly biased towards the operators that will be used in later parts in this thesis. Much much more sophisticated operators exist, but are omitted here for the sake of brevity. The example in this section should therefore not be seen as a description of the state of the art in evolutionary computation, but rather as an illustrated example for how the algorithms may look like. The interested reader is referred to [ES03, Bäc96] for broader overviews of the field.

InitialSampling(μ) – Initialization of the algorithm to sample μ points. Examples:

- Uniform Sampling: sample μ points uniformly at random (i.i.d. uniform sampling)
- Quasi-random sampling, e.g., from a low-discrepancy sequence [DP10, Mat09, Nie92]
- Latin Hypercube Designs [MBC79]

SelectC(P, k), SelectM(P, k), and Replace(P, O, k) – Selection of k points for the crossover operation, the mutation operation, and the replacement of the population, respectively. For the latter, we distinguish whether selection is applied to the (multi-)set $P \cup O$ (*plus selection*) or to O (*comma selection*). Examples:

- Uniform selection, with or without replacement
- Truncation selection: select the best k points from P (with respect to the objective function value). This operator is sometimes referred to as *elitist selection.* Different tie-breaking rules are used, e.g., random selection or diversity-maximizing selection (with respect to a diversity measure that must be specified).
- Fitness-proportional selection: select each of the k points by sampling from the distribution that assigns each $x \in P$ a probability of $f(x)/\sum_{y \in P} f(y)$.
- Tournament selection: select uniformly at random m different points in P, and keep the best one of these. Repeat the procedure k times. Instead of a uniform selection of the m "competitors", a fitness-proportional selection can be chosen. The *tournament size* m is a control parameter that needs to be chosen by the user.

Crossover(x, y) – binary variation operators. Examples:

- $\text{cross}_c(x, y)$ – Uniform crossover with bias c: for each position $i \in [1..n]$ it is decided, independently of all other decisions, whether the entry of the first argument (with probability $1 - c$) or of the second argument (with probability c) is copied. That is, the "offspring" z satisfies $\mathbb{P}[z_i = x_i] = 1 - c$ and $\mathbb{P}[z_i = y_i] = c$, for all $1 \leq i \leq n$. The by far most common setting uses $c = 1/2$, and when no further specification is made, the term "uniform selection" can be understood this way.
- k-point crossover: select $i_1, \dots, i_k$ uniformly at random and without replacement from $[1..n]$ and set $z_i \leftarrow x_i$ for $i \in [1..i_1] \cup [i_2 + 1..i_3] \cup \dots$ and set $z_i \leftarrow y_i$ for $i \in [i_1 + 1..i_2] \cup [i_3 + 1..i_4] \cup \dots$.

Mutation(x) – unary variation operators. Examples:

- $\text{flip}_k(x)$ – random search at radius k: create an offspring y by changing the entries in k pairwise different, uniformly chosen positions. k is referred to as the *mutation strength*. Different ways to change the entries in the selected positions exist. In the context of pseudo-Boolean optimization, we simply flip the bit.
- $\text{SBM}(x, p)$ – Standard bit mutation with *mutation rate* p: chooses the mutation strength k from the binomial distribution $\text{Bin}(n, p)$ and applies $\text{flip}_k(x)$. Standard bit mutation is often defined in the following, equivalent, way: decide, independently for each position $i \in [1..n]$, whether to keep the entry (with probability $1 - p$) or whether to change it (with probability p).
- $\text{SBM}_{>0}(x, p)$ – Conditional standard bit mutation with mutation rate p: choose k from the binomial distribution $\text{Bin}(n, p)$ until $k > 0$ and apply $\text{flip}_k(x)$. This is identical to sampling k from the conditional binomial distribution $\text{Bin}_{>0}(n, p)$ which re-assigns the probability to sample a 0 proportionally to all values $i \in [1..n]$. That is, $\mathbb{P}[k = i] = \binom{n}{i} p^i (1-p)^{n-i} / (1 - (1-p)^n)$
- $\text{SBM}_{0 \to 1}(x, p)$ – "Shifted" standard bit mutation with mutation rate p: choose k from the binomial distribution $\text{Bin}(n, p)$. If $k = 0$, replace it by $k = 1$. Apply $\text{flip}_k(x)$.
- Fast mutation from [DLMN17]: Sample k from the power-law distribution $\mathbb{P}[L = k] = (C^\beta_{n/2})^{-1} k^{-\beta}$ with $\beta = 1.5$ and $C^\beta_{n/2} = \sum_{i=1}^{n/2} i^{-\beta}$. Apply $\text{flip}_k(x)$.
- normal mutation from [YDB19]: sample k from the normal distribution $\mathcal{N}(pn, \sigma^2)$ and apply $\text{flip}_k(x)$. In contrast to the mutation operators discussed above, this operators allows to scale the variance of the mutation strength independently of the mean.

For the last two mutation operators, one needs to decide what to do when the chosen mutation strength k does not fall into the range $[0..n]$. Common treatments are resampling until a feasible mutation strength is obtained, capping at the boundaries, or a uniform choice of k.

Termination Criteria – Examples:

- Fixed budget B of function evaluations or iterations
- Reaching a given target value $v \in \mathbb{R}$
- No or insufficient progress in τ iterations
- Diversity falling below some threshold

These termination criteria can also be used to decide when to *restart* an algorithm. In this case, different **restart strategies** exist, for example independent restart, restart by perturbation of previously found solutions, or restart by applying some diversification strategy ("niching").

1.4 Benchmark Algorithms and Problems Frequently Appearing in this Thesis

With the examples provided above, we can now introduce two families of algorithms which play an important role in this thesis, and in the theory of IOHs in general.

Randomized Local Search (RLS): RLS is a first ascent random hill-climber. It is a (1+1) scheme, i.e., it maintains a best-so-far solution x in memory, and generates one offspring per iteration. The offspring y is created by applying the flip_k mutation operator to x, i.e., k uniformly chosen bits are flipped to generate y from x. Unless stated otherwise, we assume $k = 1$, i.e., we interpret the local neighborhood as all points at Hamming distance 1 around the current best solution. Note that RLS does not keep in mind which solutions have been visited already, nor does it track how often the entry of a given position has been flipped already. It is hence a very basic optimization strategy with – seemingly – obvious improvement potential. However, note also that it is not difficult to construct problems for which RLS is more efficient than other local search variants which try to balance the number of times each bit is flipped. For our analyses, RLS is often a good starting point, in particular when used with a fixed or with a deterministic choice of the mutation strength, as it then only has one source of randomness, not several ones.

The (1+λ) Evolutionary Algorithm (EA): Similarly to RLS, the $(1+\lambda)$ EA keeps in its memory a best so far solution x. In each iteration, λ offspring are sampled and the best one of these replaces the parent x if it is at least as good. Ties are usually broken uniformly at random for the selection among the λ offspring, and with a bias towards the offspring for the replacement of the best-so-far solution. The λ offspring are sampled via independent sampling. Each one is generated by applying the standard bit mutation operator $\text{SBM}(\cdot, p)$ to x. We will mostly focus on the case $\lambda = 1$.

Unless stated otherwise, we assume that $p = 1/n$, which is the most commonly recommended mutation rate. See [Wit13] for an example in which it is explicitly proven that this mutation rate minimizes, among all static choices for p, the expected optimization time of the $(1+1)$ EA on any linear function $f : \{0,1\}^n \to \mathbb{R}, x \mapsto \sum_{i=1}^{n} w_i x_i$.

We did not specify the termination criteria for these two algorithms, but these will be clear from the context, and are somewhat irrelevant in most of our studies, as we are typically interested in bounding the *running time* of the algorithms, i.e., the number of evaluations needed until an optimal solution is found. See Section 1.5 for a discussion of performance metrics.

We next present the two most prominent problems in the theoretical analysis of IOHs. They will also play a central role in this thesis.

OneMax: The original OneMax problem asks to optimize the function

$$\text{OM} : \{0,1\} \to [0..n], x \mapsto \sum_{i=1}^{n} x_i.$$

Of course, the optimum of this function is the all-ones string $(1, \dots, 1)$. This optimum is unique and there is no other local optimum, i.e., the problem is *unimodal*. It is not difficult to see that algorithms such as RLS and the $(1+\lambda)$ EA treat the OM function identically to any of the following instances of the generalized OneMax problem OM_z, $z \in \{0,1\}^n$:

$$\text{OM}_z : \{0,1\} \to [0..n], x \mapsto \sum_{i=1}^{n} \mathbb{1}(x_i = z_i) = |\{i \in [1..n] \mid x_i = z_i\}|, \tag{1.1}$$

where $\mathbb{1}(\mathcal{E})$ is the function that returns 1 if the event $\mathcal{E}$ is satisfied and which returns 0 otherwise. That is, $\text{OM}_z(x)$ is the number of positions in which the entries of the strings x and z agree. In this thesis, whenever we speak of OneMax, we refer to the so-generalized set of problem instances.

LeadingOnes: The original LeadingOnes problem asks to maximize the function

$$\mathrm{LO} : \{0,1\}^n \to [0..n], x \mapsto \max\{i \in [0..n] \mid \forall j \in [1..i] : x_j = 1\} = \sum_{i=1}^{n} \prod_{j=1}^{i} x_j,$$

which counts the number of initial ones. As with OneMax, we can easily generalize this problem to a permutation- and XOR-invariant class of instances by defining for each $z \in \{0,1\}^n$ and for each permutation $\sigma \in S_n$ the instance

$$\mathrm{LO}_{z,\sigma} : \{0,1\}^n \to [0..n], x \mapsto \max\{i \in [0..n] \mid \forall j \in [1..i] : x_{\sigma(j)} = z_{\sigma(j)}\}. \tag{1.2}$$

Note that $\mathrm{LO}_{z,\sigma}(x)$ is simply the maximal joint prefix of the two strings x and z, in the order induced by the permutation σ. Again it is not difficult to see that RLS and the $(1+\lambda)$ EA treat all these instances identically, in the sense that their performance trace has exactly the same distribution on any of these instances. We refer to the whole set of instances $\{\mathrm{LO}_{z,\sigma} \mid z \in \{0,1\}^n, \sigma \in S_n\}$ when speaking, in this thesis, of the LeadingOnes problem.

We briefly comment on these two problems: The OneMax function is the by far best-studied benchmark problem in the theory of IOHs, and is often referred to as the "drosophila of Evolutionary Computation"[FCSS08]. The problem has a very smooth and non-deceptive fitness landscape with a perfect fitness-distance correlation: the closer we get to the optimum, the larger the function values. Indeed, OM_z is the same problem as minimizing the Hamming distance to the *target string* z.

Differently from OneMax, the LeadingOnes instances are not *separable,* i.e., the influence of a particular bit is not independent of the other bits (for example, there is no contribution of the 1 in the string $x = (010\ldots0)$ to the function value $\mathrm{LO}(x) = 0$, whereas the contribution of this bit in the string $y = (110\ldots0)$ is one). LeadingOnes is undoubtedly the second-most studied benchmark problem in the theory of IOHs. Note that it is also a non-deceptive problem, but with a bad fitness-distance correlation: both strings $(0\ldots0)$ and $(01\ldots1)$ have function value zero, but the second one has Hamming distance 1 from the optimum, whereas the first one has Hamming distance n.

1.5 Performance Measures

While we have mentioned several times above that we are interested in studying the "performance" of IOHs, we did not yet make precise how we measure this performance. Which measure to use is indeed a highly non-trivial question, for three main reasons:

1. Unlike in classical algorithmics, where problem data is assumed to be accessible and where running times can be measured in arithmetic operations, this approach cannot be applied to IOHs, which typically spend a significant part of their time waiting for the search points to be evaluated.

2. Many IOHs are randomized algorithms, so that their performance traces are randomized as well.

3. Performance can differ between instances of the same problem, and (of course) between different problems. We therefore need to decide how to aggregate performance measures.

Luckily, the first challenge can be by-passed by simply basing all running time measures on the *number of function evaluations.* This way, we obtain performance measures that capture the main components of the running time. An advantage of this "query complexity" approach is the fact that the so-obtained running time measures are independent of the implementation of an algorithm and of the hardware on which they are run. For theoretical research, measuring function evaluations is

a very "clean" and convenient approach. Function evaluations are also the main, but not the only performance criterion regarded in empirical research [HAB+16, RT18]. However, as is the case with all complexity measures, one should not forget that there are situations in which the match between function evaluations and optimization requirements is poor, in which case other complexity measures, such as CPU times may be more appropriate, see [JZ11, JM02, WCL+14, HWHC13] for critical discussions and alternatives. Note also that by counting the function evaluations only, we may not reflect the possibility of parallelizing some of the evaluations. Where parallelization is important (and the number of parallel evaluations is not stable during the whole optimization process), other performance criteria such as the number of iterations may become more meaningful. As said, the choice of the performance measure should correspond to the specific requirements of the optimization scenario. Theoretical research, however, focuses almost exclusively on counting function evaluations or generations, and this is also the approach taken in this thesis.

Still, even if we agree to use function evaluations as our main or only performances measure, addressing the randomized nature of the performance traces of IOHs remains difficult, as their performance space is at least three-dimensional, spanned by the following criteria:

- the solution quality (*fitness*),
- the number of function evaluations (*budget*), and
- the probability of finding within a given budget of function evaluations a solution that is at least as good as a given quality threshold (*probability of success*).

In which way to aggregate this three-dimensional performance space depends again on the application and user interest. Where the number of function evaluations is the key restriction, statistics about the solution quality that can be obtained within this *fixed budget* are needed. These statistics can be some quantiles of the solution quality, or its average value. If, in contrast, a user is most interested in finding a solution of a certain quality, the number of evaluations needed to reach this *fixed target* is considered – and here again it depends on the situation whether to regard quantiles or average values. In security-related applications, but also in other settings, a high certainty to reach a minimum quality threshold is likely to be very important, whereas the probability of success is much less important in settings in which we can perform several independent optimization runs in parallel. We also note, without going further into details, that in practical applications not only the *robustness with respect to solution quality and time* matters, but also the *robustness with respect to the recommended solution.* That is, when the same or similar problem instances are solved twice, it can be important that the suggested solutions are similar, even if several solutions of (almost) identical quality exist.

In theoretical research, the by far most commonly studied metric is the *optimization time,* also referred to as the *running time* of an algorithm. The running time is the number of evaluations needed until an optimal solution is evaluated for the first time, i.e., the *first hitting time* of an optimal solution. We often bound only the expected value of this random variable, but it is becoming more and more common to also study other moments or properties of the running time.

What concerns the aggregation of performance statistics over different instances, the most common approach in the study of IOHs follows the classic approach taken in the broader Computer Science literature, which is to adopt a very cautious ("pessimistic") measure and to regard the worst case. Here again, naturally, the most suitable aggregation depends on the application at hand. Alternatives such as average-case measures or the worst-case for a $1-\varepsilon$ fraction of the instances only are subject of intensive research in the broader algorithms literature, but play – so far – only a marginal role in the theoretical analysis of IOHs. Since in this thesis we are mostly concerned with perfectly homogeneous instances/performance statistics, we can safely stick to the convention of applying a worst-case aggregation.

Summing up this discussion, we mostly focus in this thesis on *worst-case expected running time* of an algorithm A on a set of problem instances $\mathcal{F}$. Thus, formally, we regard

$$\sup_{f \in \mathcal{F}} \mathbb{E}[T(A, f)],$$

where $T(A, f)$ is the random variable that denotes the number of function evaluations that algorithm A performs on function f until it evaluates for the first time an optimal solution $x \in \mathcal{S}$. And we keep in mind the shortcomings of this measure listed above, in that by reducing the whole performance space to a single measure we necessarily lose a lot of information about (1) the anytime performance of the algorithms, (2) the distribution of the running times on a single instance, and (3) the distribution of running times across the different instances.

We should further note that theoretical research for IOHs focuses very much on *asymptotic analyses,* i.e., we typically assume that the set of problems is scalable in the dimensions, in the sense that for each (or for at least infinitely many) $n \in \mathbb{N}$ our problem is a set $\mathcal{F}_n$ of problems f that are defined over n decision variables (this is clearly the case of OneMax and LeadingOnes, as straightforwardly seen by their definitions). We are then interested in how $\sup_{f \in \mathcal{F}_n} \mathbb{E}[T(A, f)]$ scales in n.

Examples: The Running Times of RLS and the $(1+1)$ EA on OneMax and LeadingOnes. The running time of RLS on OneMax is easily seen to resemble the well-known *coupon collector problem* (see [DP09] or [Doe20a] for a gentle introduction to this problem), with the only difference being the random initialization of RLS: while the coupon collector starts with zero coupons, the initial starting point of RLS on OneMax is distributed according to a binomial distribution with n trials and success probability $1/2$. Disregarding this random starting point, RLS is easily seen to have an expected running time on OneMax of at most $n \ln(n) + \gamma n + \frac{1}{2} \approx n \ln(n) + 0.5772n$ function evaluations, where $\gamma = 0.5772\ldots$ is the Euler-Mascheroni constant. The precise complexity is $n \ln(n) + (\gamma - \ln(2))n + o(1) \approx n \ln n - 0.1159n$, by a result that we have shown in [DD16a]. We will see in Sections 2.4.2.1 and 3.2 that these bounds can be improved by adjusting the mutation strength during the optimization process.

The $(1 + 1)$ EA has a constant overhead and achieves an expected running time on OneMax of $(1 - o(1))en \ln(n)$, see [HW19, HPR$^+$18] for a much more precise bound. While the upper bound is rather straightforward, the lower bound is surprisingly difficult to prove, since – in contrast to RLS – the algorithm could, and frequently does, make progress of more than just one.

LeadingOnes was originally designed in [Rud97] to disprove a previous conjecture of Mühlenbein [Müh92], who claimed that the expected running time of the $(1 + 1)$ EA on every unimodal function is $O(n \log n)$. While Rudolph showed experimentally that its expected running time is $\Theta(n^2)$, this bound was formally proven a bit later in [DJW02]. Exact expressions for the expected running time of the $(1 + 1)$ EA on LeadingOnes are available in [BDN10, Sud13, Lad05] (see also Section 4.3). It is not difficult to see that RLS has an expected running time of $(1 + o(1))n^2/2$ on LeadingOnes.

1.6 Outline of the Thesis

Undoubtedly, the vast majority of research activities on IOHs aims at designing efficient solvers or at methods that help one choose a suitable heuristic for a given optimization task. A substantial fraction of my own work also falls into this category. Another major theme in my research, however, has always been on studying the *limits* of sampling-based optimization, commonly referred to in the evolutionary computation community as **black-box complexity**. Results on this topic will be presented in Chapter 2. In Chapter 3 we summarize two examples that illustrate how such black-box complexity results can inspire the **design of new algorithms.** More precisely, we illustrate in this section the key observations that let us to design the $(1 + (\lambda, \lambda))$ GA presented in [DDE15] and to the learning-based parameter control scheme presented in [DDY16]. **Parameter control,** that is, the dynamic configuration of IOHs, is also the topic of Chapter 4. A few selected results not covered by the above topics will be briefly highlighted in Section 5. We conclude this thesis in Chapter 6 by providing an outlook on what I consider to be important directions for future research.

Chapter 2

Black-Box Complexity Theory Continued

Whereas research of IOHs very clearly aims to contribute to designing algorithms that most efficiently optimize a given problem $f : \mathcal{S} \to \mathbb{R}$, it can be very instructive to know where the limits of sampling-based optimization techniques are. Quantifying the performance of a (possibly not explicitly known) best solver is the subject of *black-box complexity*. Black-box complexity has been the central topic of my PhD thesis [Win11]. It continues to play an important role in my research activities since. In the following sections, I highlight a few selected results obtained in the last years. A more exhaustive summary of existing black-box complexity results can be found in my book chapter [Doe20b].

2.1 Black-Box Complexity – Definition

Recalling that we measure the performance of sampling-based heuristics by the number of function evaluations, this criterion must be reflected in a matching complexity theory. Following the discussion in Section 1.5, we are thus mostly interested in studying, for a given collection of problem instances $\mathcal{F}$, the best possible (across all algorithms) worst-case (with respect to the problem instances in $\mathcal{F}$) average (over all independent runs of the algorithm) running time that an algorithm can achieve. That is, we define the black-box complexity of a class $\mathcal{F}$ of problem instances as

$$\mathrm{BBC}(\mathcal{F}) := \inf_{A} \sup_{f \in \mathcal{F}} \mathbb{E}[T(A, f)], \tag{2.1}$$

where we recall that by $T(A, f)$ we denote the number of function evaluations that algorithm A performs on problem instance f until it evaluates for the first time an optimal solution.

The definition above does not make any restriction on the type of algorithm, or the knowledge that it uses about the problem instances in $\mathcal{F}$. This, in particular, implies that the black-box complexity of a single function $\mathcal{F} = \{f\}$ is always one: the algorithm querying an optimal solution $\arg\max f$ in the first iteration achieves this bound. It therefore only makes sense to study the black-box complexity of collections $\mathcal{F}$ which comprise several functions or, as we shall discuss below, to restrict the set of algorithms under consideration. In practice, we often seek to find problems that are structurally equivalent or at least similar to a problem of interest. Here, the definition of "structurally equivalent" can be highly subjective. A common way to generalize a function f is by concatenating it with automorphisms of its domain [RV11, LW12, DKLW13]. For example, in the case of pseudo-Boolean

functions $f : \{0,1\}^n \to \mathbb{R}$, and the domain being understood as the Hamming cube (i.e., we consider as neighbors any two points that differ in exactly one position), we may consider the class

$$\mathcal{F}(f) = \{f_{z,\sigma} \mid z \in \{0,1\}^n, \sigma \in S_n\} \tag{2.2}$$

with

$$f_{z,\sigma}(x) := f(\sigma(x_1 \oplus z_1, \dots, x_n \oplus z_n)) = f(x_{\sigma(1)} \oplus z_{\sigma(1)}, \dots, x_{\sigma(n)} \oplus z_{\sigma(n)}). \tag{2.3}$$

As we have discussed in Section 1.4, several classical IOHs treat all such functions equally, in the sense that their performance will be identical on each instance $f_{z,\sigma}$.

Since invariance with respect to the bit positions is not always desirable (e.g., when the order of decision variables corresponds to interactions between them), another generalization $\mathcal{F}^* \subseteq \mathcal{F}$ commonly studied comprise only those instances in $\mathcal{F}$ for which σ is the identity; i.e.,

$$\mathcal{F}^*(f) = \{f_z \mid z \in \{0,1\}^n\} \text{ with } f_z(x) = f(x_1 \oplus z_1, \dots, x_n \oplus z_n).$$

A second, complementary way of obtaining meaningful complexity statements is to restrict the class of algorithms under investigation. That is, for a given collection $\mathcal{A}$ of algorithms, we define the *$\mathcal{A}$-black-box complexity of $\mathcal{F}$* (or, likewise, the *$\mathcal{A}$-restricted black-box complexity of $\mathcal{F}$*) as

$$\mathrm{BBC}(\mathcal{F}, \mathcal{A}) := \inf_{A \in \mathcal{A}} \sup_{f \in \mathcal{F}} \mathbb{E}[T(A, f)]. \tag{2.4}$$

When no assumption on $\mathcal{A}$ is made, we speak of the *unrestricted black-box model* and the *unrestricted black-box complexity*, respectively.

Warning: Most black-box complexity models do not make any assumption on the (wall-clock) time that an algorithm requires between any two queries. The complexity bounds are therefore not comparable to classical complexity notions. In particular, NP-hard problems can have very small black-box complexity, as it essentially suffices to learn the problem instance in a sufficient (often, linear) number of queries, and then computing an optimal solution *offline*, i.e., without further queries. The last step may – in the extreme case – comprise an exhaustive search for an optimal solution; see [DJW06, DDK14] for examples in the unrestricted and the so-called unbiased black-box models, respectively. Most of the upper bounds mentioned below, however, can be obtained by polynomial-time algorithms.

2.2 Black-Box Complexity – Motivation

As is the case for many sub-domains in Theoretical Computer Science, a core motivation to study black-box complexity models is to further our understanding of the gaps between state-of-the-art techniques and their theoretical performance limits, with the objective to derive insights that will eventually help us design more efficient solution techniques. Black-box complexity contributes to this goal in several complementary ways:

(1) A first benefit of black-box complexity is that it enables the above-mentioned **evaluation of how well we have understood a black-box optimization problem, and how suitable the state-of-the-art heuristics are.** Where large gaps between the black-box complexity and the performance of a best known solver exist, we may want to explore alternative algorithmic solutions, in the hope to identify more efficient solvers. Likewise, we can stop striving for more efficient algorithms when the two quantities match (or are close to each other).

(2) Another advantage of black-box complexity studies is that they **allow to investigate how certain algorithmic choices influence the performance:** By restricting the class of algorithms under consideration, we can judge how these restrictions increase the complexity of a black-box optimization problem. In the context of evolutionary computation, interesting restrictions include the amount of memory that is available to the algorithms [DJW06, DW14a], the number of solutions that are sampled in every iteration [BLS14, LS19], the distributions from which solution candidates are generated [LW12, RV11], the selection principles according to which it is decided which search points to keep for future reference [DL17a], etc. Comparing the black-box complexity $\mathrm{BBC}(\mathcal{F}, \mathcal{A})$ with $\mathrm{BBC}(\mathcal{F}, \mathcal{B})$ for two collections $\mathcal{B} \subseteq \mathcal{A}$ quantifies the performance loss caused by restricting our attention to the smaller class of algorithms. For example, by restricting the memory available to the algorithms under consideration, we can analyze the effects of not storing the set of *all* previously evaluated solution candidates, but only a (possibly small) subset thereof.

(3) As we shall see below, the black-box complexity of a problem can be significantly smaller than the performance of a best known "standard" heuristic. In such cases, the small complexity is often attained by a very problem-tailored black-box algorithm, which is not representative for common sampling-based optimization heuristics. Interestingly, it turns out that we can nevertheless learn from such highly specific algorithms, as they often incorporate ideas that can be beneficial much beyond the particular problem at hand. As we shall demonstrate in Chapter 3, even for very well-researched optimization problems, such ideas can give rise to the design of novel heuristics which are provably more efficient than standard solutions. This way, black-box complexity serves as a **source of inspiration for the development of novel algorithmic ideas that lead to the design of better search heuristics.**

2.3 Results for the Unrestricted Model

We summarize in this section two results for the unrestricted black-box model. These results were proven in [DDST16] and [AAD+19], respectively. In particular the first result enjoys some popularity outside the evolutionary computation community, since it improved a 30 years old bound on a popular board game. The second result is mostly interesting from a methodological point of view, since both the upper and the lower bounds proven to obtain the tight black-box complexity of the LeadingOnes problem are highly non-trivial.

2.3.1 OneMax and Mastermind

One of the most classic black-box complexity results – albeit formulated in a different context (as a coin-weighing problem) – goes back to Erdős and Rényi, who showed in [ER63] that the unrestricted black-box complexity of the generalized OneMax problem class defined in 1.1 is $\Theta(n/\log n)$. More precisely, Erdős and Rényi showed that the unrestricted black-box complexity of OneMax is at least $(1 - o(1))n/\log_2(n)$ and at most $(1 + o(1))\log_2(9)n/\log_2(n)$. The upper bound was improved to $(1 + o(1))2n/\log_2(n)$ in [Lin64, Lin65, CM66]. To date, the factor two gap between upper and lower bound remains open. The algorithm certifying the upper bound is based on pure random sampling and an expensive offline computation of the still possible target strings z. That the $(1 + o(1))2n/\log_2(n)$ upper bound can also be achieved by a proper polynomial-time algorithm has been shown in [Bsh09].

The result of Erdős and Rényi was later generalized by Chvátal to larger alphabet sizes. Chvátal [Chv83] showed that (in our terminology) the unrestricted black-box complexity of the class

$$f_z : [0..k-1]^n \to \mathbb{R}, x \mapsto |\{i \in [1..n] \mid x_i = z_i\}|$$

of "Mastermind" problems with n "positions" and k "colors" is $\Omega(n \log k / \log n)$ and that for $\varepsilon > 0$ and $k \leq n^{1-\varepsilon}$, it is at most $(2+\varepsilon)n(1+2\log k)/\log(n/k)$. The name of the Mastermind problem goes back to a popular board game from the seventies and eighties of the last century, where a *codemaker* selects a secret color code z and the *codebreaker* aims to identify the code in as few queries as possible. Each query is answered by the number of positions in which codebreaker's code agrees with codemaker's. See Figure 2.1 for a picture of this game.

Note that for $k \leq n^{1-\varepsilon}$, $\varepsilon > 0$ being a constant, Chvátal's result gives an asymptotically tight bound of $\Theta(n \log k / \log n)$. Similarly to the random guessing strategy by Erdős and Rényi, it is sufficient to query this many *random* queries, chosen independently and uniformly at random from $[0..k-1]^n$. With high probability, the scores received for these guesses reduce the set of possible target strings z to a single solution (which is then also the optimum of the function). Note that no adaptation is needed for such combinations of n and k to *learn* the secret target vector z (but, of course, we need adaptation to *query* this optimum).



Figure 2.1: The commercial Mastermind game, a k-color extension of the OneMax problem.

This situation changes for the regime around $k = n$, which has been the focus of several subsequent works [CCH96, Goo09, JP11]. These works all show bounds of order $n \log n$ for the $k = n$ Mastermind problem. In [DDST16] we could improve these bounds to $O(n \log \log n)$, which is to date the best known upper bound for the unrestricted black-box complexity of the n-color, n-position Mastermind game.[1] The best known lower bound for this problem is the linear one reported in [Chv83], which is straightforwardly proven using a basic information-theoretic argument: essentially, each query reveals at most $\log(n+1)$ bits of information (since each function value is an integer between 0 and n). Since we need to learn $n \log n$ bits of information in total (this is the smallest amount of bits needed to describe the secret code z), we easily derive that we need at least n queries to obtain the secret code. This informal argument can be be made precise using Yao's minimax theorem [Yao77]. Essentially, the latter enables us to obtain lower bounds for randomized algorithms by studying the performance of deterministic algorithms on randomly chosen problem instances. Since the class of deterministic algorithms is much smaller than that of all randomized ones, proving complexity bounds for deterministic algorithms can be substantially easier than directly analyzing randomized ones.

Theorem 2.3.1 (Theorems 2.1 and 2.3 in [DDST16])**.** *For Mastermind with n positions and $k = \Omega(n)$ colors, the unrestricted black-box complexity of the n-position, k-color Mastermind game is $O(n \log \log n + k)$. For $k = o(n)$ it is $O\left(n \log\left(\frac{\log n}{\log(n/k)}\right)\right)$. This bound can be realized by a polynomial-time deterministic winning strategy.*

The two central ideas in the proof of Theorem 2.3.1 are (1) the observation that a score $f_z(x) = 0$ is easy to interpret, as it allows to remove for each position i the color x_i from the set C_i of still possible colors for that position, and (2) the idea that we can divide the string into small blocks, for which we can efficiently learn which blocks do not contribute to the overall score (as opposed to learning this for individual positions). That is, we only try to find blocks which contributes 0 to $f_z(x)$, and then reduce the set C_i for all the positions i in that block. The $O(n \log \log n)$ strategy used to prove Theorem 2.3.1

[1] We note, without further going into the details, that results for other combinations of n and k, as well as for the case with so-called "white pegs" are available in [DDST16]. In the original Mastermind game, the $f_z(x)$ score is indicated by black pegs (red in Fig. 2.1), whereas the white pegs indicate the number of correct colors in a non-agreeing position (i.e., formally the number of white pegs is defined as $\max_{\rho \in S_n} |\{i \in [1..n] \mid z_i = x_{\rho(i)}\}| - f_z(x)$). The case $k = n$ and "black-pegs only", however, is the most interesting one, as argued in detail in [DDST16].

alternates between random guessing of strings in $C_1 \times \ldots \times C_n$, and the identification of the "0-blocks". The size of the blocks shrinks as the search progresses, reflecting the fact that the probability that a given position i does not contribute to the score decreases with decreasing size $|C_i|$. The problem of identifying which blocks contribute zero to the score is reduced to a coin weighing problem previously studied in [GK00, Bsh09], in which one is given m coins of unknown integer weights and the goal is to identify the weight of every coin with as few weighings as possible. A key challenge in analyzing this strategy is in ensuring that the sizes of the sets C_i do not become imbalanced. To by-pass this challenge, the algorithm proceeds in phases, and in each phase we never reduce the size of the sets C_i below a certain threshold. When the size of the blocks (and of the C_i, respectively) becomes to small, the algorithm switches to the random guessing strategy of Chvátal [Chv83]. Overall, the proof of Theorem 2.3.1 is very heavily inspired by previous black-box complexity results in which we had already applied the idea to divide the string into smaller blocks, see [DW14c, DW14a, DW14b] for explicit examples.

Coming back to the discussion of *adaptivity,* we also showed in [DDST16] that in the $k = \Theta(n)$ regime any $o(n \log n)$ algorithm has to be *adaptive,* i.e., the secret code cannot be correctly guessed with fewer than $o(n \log n)$ guesses if these have to be decided upon prior to the first evaluation. This lower bound can be proven by an entropy-compression argument, similar to the one presented in [MT10].

Theorem 2.3.2 (Theorems 4.1 and 4.3 in [DDST16])**.** *The non-adaptive unrestricted black-box complexity of the Mastermind problem with n positions and k colors is $\Omega\left(\frac{n \log k}{\max\{\log(n/k),1\}}\right)$. For $k \leq n$ this bound is tight and can be achieved by a deterministic guessing strategy. In particular, the non-adaptive unrestricted black-box complexity of the Mastermind problem with n positions and n colors is $\Theta(n \log n)$.*

2.3.2 LeadingOnes

We now turn our attention to the unrestricted black-box complexity of the LeadingOnes problem $\mathcal{F}(\mathrm{LO}) = \{\mathrm{LO}_{z,\sigma} \mid z \in \{0,1\}^n, \sigma \in S_n\}$ of instances $\mathrm{LO}_{z,\sigma}$ as defined in 1.2.

To gain some intuition about the problems in $\mathcal{F}(\mathrm{LO})$, let us briefly consider the black-box complexity of the smaller set $\mathcal{F}^*(\mathrm{LO})$, in which no re-ordering of the bit positions is enforced and we only need to learn the target string z. In this case, a function value of $k \in [0..n]$ indicates that the entries in the first k positions are correct, whereas the $(k+1)$-st entry is incorrect. We cannot infer any information about the positions in the "tail" $(k+2, \ldots, n)$. Therefore, an efficient optimization strategy is to keep in the next iterations the first k bits unchanged, while flipping the $(k+1)$-st. It does not matter what we do in the tail, as we have no information about the correctness of these bits. Assuming a random problem instance, the expected function value of the so-generated point is $k + 1 + \sum_{j=k+2}^{n} 2^{k+1-j}$. This simple algorithm therefore optimizes any instance in $\mathcal{F}^*(\mathrm{LO})$ using at most $n/2 + o(n)$ function evaluations on average. Using Yao's minimax principle [Yao77], it is not difficult to show that this bound is tight, i.e., it holds that $\mathrm{BBC}(\mathcal{F}^*(\mathrm{LO})) = n/2 \pm o(n)$. Both bounds were proven in [DJW06].

When moving on to optimizing instances of the broader class $\mathcal{F}(\mathrm{LO})$, we lose information about the order of the positions. That is, while a score of k still tells us that the "first" (according to the unknown permutation σ) bits are correct, and that the $(k+1)$-st one is incorrect, we do not know in this setting – *a priori* – where these positions are. For efficiently optimizing a function $\mathrm{LO}_{z,\sigma}$, we cannot avoid to learn some information about the permutation σ. A simple strategy to do so would be to apply a sequential binary search, which learns the positions $\sigma(1), \sigma(2), \ldots$ one after the other. This strategy is easily seen to yield an order $n \log n$ algorithm. This, however, is not the most efficient way of optimizing the instances $\mathrm{LO}_{z,\sigma}$, as we had already proven in [DW12]. More precisely, we developed in [DW12] a strategy which optimizes any instance from $\mathcal{F}(\mathrm{LO})$ using at most $O(n \log n / \log \log n)$ queries on average. In [AAD+19] we could improve this bound to $O(n \log \log n)$ by optimizing the way

in which we interleave the binary search learning procedure with steps that capitalize on the partial knowledge gained about the problem instance (and in particular about the permutation σ) at hand. We furthermore also showed that our algorithm is asymptotically optimal.

Theorem 2.3.3 (Theorems 4.1 and 5.1 in [AAD+19])**.** *The unrestricted black-box complexity of* $\textsc{LeadingOnes}_n$ *is* $\Theta(n \log \log n)$*. The upper bound can be achieved by a polynomial-time algorithm.*

The proofs of the upper and lower bounds are both rather involved. For the lower bound, Yao's minimax principle is applied to the uniform distribution over the instances $\mathrm{LO}_{z,\sigma}$ with $z_{\sigma(i)} := (i \bmod 2), i = 1, \ldots, n$, indicating that the complexity of the instances $\mathrm{LO}_{z,\sigma}$ originates indeed in the difficulty of learning the permutation σ (since σ determines the optimum).

In contrast to the situation for Mastermind (Theorem 2.3.1), the algorithm used to prove the upper bound in Theorem 2.3.3 cannot be derandomized. In fact, it is not difficult to show that for any deterministic algorithm there exists an instance $\mathrm{LO}_{z,\sigma} \in \mathcal{F}(\mathrm{LO})$ enforcing $\Omega(n \log n)$ function evaluations until the optimum is queried for the first time.

Theorem 2.3.4 (Theorem 3.1 in [AAD+19])**.** *The deterministic black-box complexity of* $\mathcal{F}(\mathrm{LO})$ *is* $\Theta(n \log n)$*, and hence asymptotically larger than its randomized counterpart from Theorem 2.3.3. The above-mentioned binary search strategy is therefore asymptotically optimal among all deterministic algorithms (in the big-Oh sense).*

2.4 Results for Restricted Black-Box Models

After having considered the unrestricted black-box model in the previous section, we next focus on *restricted black-box models,* in which the class of admissible algorithms can be substantially smaller than in the unrestricted case. In Section 2.4.1 we first discuss the effects of combining two previously studied black-box models, the memory-restricted one suggested in [DJW06] and the ranking-based one introduced in [DW14b]. In Section 2.4.2 we then discuss new results for the *unbiased black-box model*, a concept which revived black-box complexity around ten years ago, with the conference version [LW10], which later appeared as full version in [LW12]. In nutshell, the unbiased black-box model restricts the class of algorithms to those that are invariant under the Hamming automorphisms described in Section 2.1. We finally introduce in Section 2.4.3 the *elitist black-box model*, a restricted black-box model that we have introduced in [DL17a] to analyze the impact of the selection behavior of IOHs. Intuitively, this model requires that only the μ best-so-far solutions can be kept in the memory.

2.4.1 The Combined Memory-Restricted Ranking-Based Black-Box Model

Two key results of my PhD thesis showed that the (1+1) memory-restricted black-box complexity of the $\textsc{OneMax}$ problem, as well as its ranking-based black-box complexity are $\Theta(n/\log n)$, and thus of the same asymptotic order as its unrestricted black-box complexity, which we have already discussed in Section 2.3.1. These results were proven in [DW14a] and [DW14b], respectively. In a nutshell, algorithms covered by the $(1 + 1)$ memory-restricted black-box model can only store one previously sampled solution and its objective value. They cannot store any other information about the optimization process, and not even an iteration counter. Ranking-based algorithms, on the other hand, do not have access to the absolute function values, but only to the ranking of the points induced by their function values.

While the impact of memory-restriction and ranking-basedness had previously been studied in isolation, our work [DL17b] was the first to study the effects of combining these two restrictions. It is not difficult to see, by the standard information-theoretic argument already presented above, that the

(1+1) memory-restricted ranking-based black-box complexity of OneMax is at least linear in n, and hence asymptotically larger than the pure ranking-based and the pure memory-restricted black-box complexity. However, in [DL17b] we also showed lower bounds for the combined model which are by a constant factor stronger than the simple information-theoretic ones. These bounds are thus stronger than any bound obtained by reducing the combined model to an existing black-box model with a single restriction. We refer the interested reader to Theorem 2 in [DL17b] for the details.

On the more constructive side, we showed in [DL17b] that the linear lower bound for the (1+1) memory-restricted ranking-based black-box complexity of OneMax is asymptotically tight, i.e., we presented an algorithm which meets all the requirements of the combined black-box model and which optimizes any OneMax instance using, on average, at most a linear number of queries.

Theorem 2.4.1 (Corollary 1 in [DL17b])**.** *The (1+1) memory-restricted, ranking-based black-box complexity of the* OneMax *problem is* $\Theta(n)$.

The algorithm developed to prove Theorem 2.4.1 is far from being straightforward, given that the previously best known algorithms for the single-restriction models all very crucially violate the restriction implied by the other. That is, the previously studied memory-restricted algorithms make use of absolute fitness values, whereas the query-efficient ranking-based algorithms rely on access to a large number of previously sampled points. A different strategy was therefore needed to derive the linear black-box algorithm which satisfies both requirements at the same time. We could nevertheless exploit ideas from previous black-box complexity studies. Notably, we reserve some parts of the solution for implementing an iteration counter. Implementing this counter is one of the most tricky parts in the proof of Theorem 2.4.1.

We also extended Theorem 2.4.1 to the $(\mu + \lambda)$ memory-restricted ranking-based black-box model, in which the algorithms may store up to μ previously sampled search points and can sample up to λ points per iteration. More precisely, we show that also in these cases the information-theoretic lower bound is matched by an algorithm satisfying all the requirements.

Theorem 2.4.2 (Corollary 1 in [DL17b])**.** *For* $1 < \lambda < 2^{n^{1-\varepsilon}}$, $\varepsilon > 0$ *being an arbitrary constant, the* $(1 + \lambda)$ *memory-restricted ranking-based black-box complexity of* OneMax *is* $\Theta(n/\log \lambda)$ *(in terms of generations), while for* $\mu = \omega(\log^2(n)/\log\log n)$ *its* $(\mu+1)$ *memory-restricted ranking-based black-box complexity is* $\Theta(n/\log \mu)$.

We will come back to these results in Section 2.4.3.1, when we discuss the elitist black-box model.

2.4.2 The Unbiased Black-Box Model

As mentioned above, the unbiased black-box model was introduced in [LW10, LW12]. It restricts the distributions from which the algorithms may sample new solution candidates. These have to be unbiased with respect to automorphisms of the hypercube. In addition to requiring invariance with respect to Hamming automorphisms, the unbiased black-box model also allows to discriminate between algorithms of different *arity*, where here in this context the arity of a sampling distribution is measured by the number of points that are needed to specify it. A *k-ary unbiased operator* is hence an operator sampling from a family $(D(\cdot \mid x^1, \dots, x^k))_{(x^1,\dots,x^k)\in\{0,1\}^n\times\dots\times\{0,1\}^n}$ of distributions that each satisfy the following two conditions:

(i) $\forall z \in \{0,1\}^n \, \forall y \in \{0,1\}^n : D(y \mid x^1, \dots, x^k) = D(y \oplus z \mid x^1 \oplus z, \dots, x^k \oplus z)$, (XOR invariance)

(ii) $\forall \sigma \in S_n \, \forall y \in \{0,1\}^n : D(y \mid x^1, \dots, x^k) = D(\sigma(y) \mid \sigma(x^1), \dots, \sigma(x^k))$, (permutation invariance)

where we recall from Section 1.3 that we abbreviate by S_n the set of all permutations of the index set $[1..n]$ and by $\sigma(x)$ we denote the reordered string $(x_{\sigma(1)}, \dots, x_{\sigma(n)})$. Intuitively speaking, an unbiased operator does not discriminate between the bit positions nor between the bit values. Concretely, these operators can be described as follows:

- Uniform random sampling is a 0-ary unbiased operator. No other 0-ary unbiased operator exists.
- A unary unbiased operators takes as input one search point and specifies a distribution for the radius $r \in [0..n]$ of the Hamming sphere at which the output is sampled. Once r is fixed, all points at Hamming distance r from the input are equally likely to be chosen, i.e., the offspring is obtained by applying the flip_r operator. This characterization has been formally proven in [DDY20], but – as noted there – it can (with some effort) also be derived from [DKLW13, Proposition 19].

  As mentioned in Section 1.1, the context of evolutionary computation, unary operators are referred to as *mutation operators*. All mutation operators listed in the example of Section 1.1 are unary unbiased operators.
- Binary and higher-arity unbiased operators take as input two or more points and specify (through a possibly randomized process, as described for the unary case) for each of them at which radius the eligible samples are. The output is sampled uniformly at random from the set of points satisfying all these constraints. Uniform crossover and majority vote are classical examples for such operators. The k-point crossover operators, however, are not unbiased, as they are not invariant with respect to a rearrangement of the bit position.

A k-ary unbiased algorithm can now be defined as an algorithm that only uses unbiased operators of arity at most k. The k-ary unbiased black-box complexity of a class $\mathcal{F}$ is the complexity of $\mathcal{F}$ with respect to all these algorithms. Note that algorithms that are unbiased according to the definitions above treat all the instances $f_{z,\sigma}$ in the set $\mathcal{F}(f)$ (as introduced in Equations (2.2) and (2.3)) identically. For the unbiased black-box model, it therefore suffices to consider a single instance of a so-defined problem class.

The unbiased black-box model described above was generalized to other search spaces in [RV11, DKLW13]. Details are omitted here in the interest of space.

In the remainder of this section we present two results obtained for the unbiased black-box model. The first one, presented in Section 2.4.2.1, provides a precise bound for the unary unbiased black-box complexity of OneMax. The second result, focus of Section 2.4.2.2, analyzes how the k-ary unbiased black-box complexity of so-called "jump" functions depends on the jump size ℓ and the arity k. The results are based on the journal papers [DDY20] and [DDK15], respectively.

2.4.2.1 The Unary Unbiased Black-Box Complexity of OneMax

We recall from Section 2.3.1 that the unrestricted black-box complexity of OneMax is known to be between $(1 \pm o(1))n/\log n$ and $(2 \pm o(1))n/\log n$. Already in [LW12] it was shown that this running time cannot be achieved by unary unbiased algorithms. More precisely, Lehre and Witt proved that the unary unbiased black-box complexity of OneMax is $\Theta(n \log n)$. The upper bound is matched by standard IOHs such as RLS and the (1+1) EA. The lower bound was shown to hold for all functions with a unique global optimum. This was proven via a potential function argument, or, more precisely, using a multiplicative drift theorem for lower bounds, cf. Theorems 5 and 6 in [LW12] for details. In [DDY20] we extended the result from [LW12] by making precise not only the leading constant, but also the first lower order term of the unary unbiased black-box complexity of OneMax.

Theorem 2.4.3 (Theorem 37 in [DDY20])**.** *The unary unbiased black-box complexity of* OneMax *is $n \ln n - cn \pm o(n)$ for a constant c between 0.2539 and 0.2665. The constant c can be numerically computed to arbitrary precision by the approach described in [DDY20, Section 5]. The lower bound extends to all unimodal functions; i.e., for any $f : \{0,1\}^n \to \mathbb{R}$ with unique global optimum, the unary unbiased black-box complexity of $\{f\}$ is at least $n \ln n - cn \pm o(n)$, for the same constant c as above.*

To put the result presented in 2.4.3 into context, we note that it is in line with a major quest in the field to obtain performance guarantees that are more precise than just determining the order of magnitude. That is, where "big-Oh" asymptotic results have long dominated the field, it has become much more common in recent years to specify leading constants or even lower-order terms [HW19, HPR+18, Wit13, DD16a, CSWA15, dPdLDD15]. Such bounds do not only offer much more accurate performance guarantees, but they also help us improve our methodological skills and, not less important, they allow us analyze the influence of the control parameters, such as the mutation rate [Wit13], or the offspring population size (see Theorem 5.4.1) at a much higher precision. Our result summarized in Theorem 2.4.3 shows that this precision gain in running time analysis can be extended to black-box complexity.

To prove the upper bound in Theorem 2.4.3 we showed that the *drift-maximizing* algorithm, which in each iteration takes a best-so-far solution and mutates it by flipping that number of bits which maximizes the expected fitness, is almost optimal. To define this algorithm more precisely, we recall from our discussion above that a unary unbiased algorithm can only use random sampling or unary unbiased operators. Each unary unbiased operator is characterized by a probability distribution over the possible distance $[0..n]$ at which the offspring are sampled. The drift-maximizing algorithm introduced in [DDY20] uses in each step a deterministic radius r. This radius r is determined by maximizing the expression $\mathbb{E}_{y\sim\mathrm{flip}_r(x)}[\mathrm{OM}(y)]$, where x is a current-best solution. Put differently, r is the value which maximizes the average of the values in the multi-set $\{\mathrm{OM}(y) \mid y \in \{0,1\}^n, H(x,y) = r\}$ (where $H(x,y)$ is the Hamming distance between x and y).

Theorem 2.4.4 (Theorem 11 in [DDY20])**.** *Let $T(\mathit{driftmax}, \mathrm{OM})$ be the expected running time of the drift maximizing algorithm on* OneMax*. Let $UBBC_1(\mathrm{OM})$ be its unary unbiased black-box complexity. Then $\mathbb{E}[T(\mathit{driftmax}, \mathrm{OM})] - uBBC_1(\mathrm{OM}) = \Theta(n^{2/3} \ln^9 n)$.*

To understand the gain of the drift-maximizing algorithm over previously studied unary unbiased algorithms, we can rewrite the expression in Theorem 2.4.3 to $n\,(\ln(n/3) + \gamma + c') + o(n)$, where now c' is a constant between 0.2549 and 0.2675. The previously best known unary unbiased algorithm was a variant of RLS which uses a best-of-μ initialization, for a suitably chosen μ. We have presented this algorithm in [dPdLDD15]. It has an expected runtime equaling that of RLS up to an additive term of order $o(n)$. It is hence $n(\ln(n/2) + \gamma) \pm o(n)$. For sufficiently small $\varepsilon > 0$, the drift-maximizing algorithm derived in [DDY20] is thus by an additive $(\ln(3) - \ln(2) - c')n \pm o(n)$ term faster, on average, than RLS or the algorithm presented and analyzed in [dPdLDD15]. That is, compared to these two algorithms, the drift maximizing algorithm saves between $0.138n \pm o(n)$ and $0.151n \pm o(n)$ iterations on average. For concrete problem dimensions n up to 10,000, the drift-maximizing mutation strengths r were computed in [BD19].

The above-mentioned results come with a number of important insights into the structure of the OneMax problem. For example, we have shown that flipping an even number of bits cannot maximize the expected progress. We also showed that the drift-maximizing number of bits to flip decreases monotonically with decreasing distance to the optimum. Finally, we extended our results to a *fixed-budget setting*, where we bounded the expected solution quality that an optimal unary unbiased black-box algorithm can achieve within a given budget of function evaluations (see Section 3.2 for more details).

An important insight from our work is the observation that a dynamic choice of the mutation strength can be beneficial even for simple optimization problems like OneMax. We will revisit this topic in Section 3.2 and in Chapter 4, where we will discuss how this work has influenced the development of efficient parameter control techniques. On the methodological side, we introduced for the proof of Theorem 2.4.3 new versions of the lower bound variable drift theorem which can tolerate large progresses if these happen with sufficiently small probability. This situation, which may occur in the optimization of OneMax, were not well covered by the previous drift theorems.

From a high-level point of view, we note that Theorem 2.4.4 essentially confirms the intuition that greedily maximizing the step-wise progress is an efficient solution strategy for problems with a good

fitness-distance correlation (as mentioned in Section 1.4, OneMax has a perfect correlation between function values and distance to the optimum). This is a core design principle used in many evolutionary algorithms. Clearly, it is not hard to see that drift-maximization is not optimal for each and every problem, as can be easily demonstrated by examples where the fitness leads the algorithm into local optima far away from the optimal solution [DJW02]. Other examples where drift-maximization fails are the difficult-to-optimize monotonic functions constructed in [DJS+13, LS18, Len18], where the fitness leads to the optimum, but via a prohibitively long trajectory. Note also that while our theorem confirms that drift-maximization is *almost* optimal for OneMax, it is also not very difficult to prove that it is not *strictly* optimal. Indeed, our exact numerical computation of the black-box complexity in [BD19] showed that the best unary unbiased black-box algorithm can be faster than the drift-maximizer. The differences, however, are very small: for all numerically evaluated dimensions $n \leq 10,000$ the difference in the expected runtime was less than one iteration. In [BD20], we have extended this non-optimality result for the drift maximizing algorithm to $(1+\lambda)$ EAs.

2.4.2.2 The Unbiased Black-Box Complexity of Jump Functions

The result from [LW12] that any unimodal function has an $\Omega(n \log n)$ unary unbiased black-box complexity implies that a unary unbiased algorithm cannot intentionally sample the optimum even if it knows exactly where it is located. The requirement to sample only from unary unbiased distributions is thus quite restricting (recall here that the unrestricted black-box complexity for such a case would be one, as the algorithm could simply query the optimum in the first evaluation). However, some lower order improvements over RLS are possible, as we have shown in Theorem 2.4.4. These improvements were possible because the algorithm knows at each step how far it is away from the optimum, due to the perfect fitness-distance correlation of the OneMax function. It could set the mutation strength r accordingly. In the study leading to journal paper [DDK15], we wanted to know how much the efficiency suffers if the algorithm cannot immediately infer from the function values how far it is away from the optimum. We obtain such a situation by blanking out the full ℓ-neighborhood of the optimum. For symmetry reasons, we need to do the same around its complement, for we could otherwise direct the algorithm there and then apply the unary unbiased $\text{flip}_n(\cdot)$ operator, which would deterministically give us the optimal solution. That is, we considered in [DDK15] the functions

$$\textsc{Jump}_\ell : \{0,1\}^n \to [0..n], x \mapsto \begin{cases} n, & \text{if } |x|_1 = n; \\ |x|_1, & \text{if } \ell < |x|_1 < n-\ell; \\ 0, & \text{otherwise,} \end{cases}$$

where $|x|_1 := \text{OM}(x) := \sum_{i=1}^n x_i$ denotes the number of ones in x. Jump functions have a long history in the theory of evolutionary algorithms, starting with the seminal paper [DJW02]. A number of different variants have been studied since, see [DDK15] for a short discussion and references.

When setting $\ell = n/2 - 1$ (and assuming an even value of n), only the unique optimum $(1, \dots, 1)$ of $\textsc{Jump}_\ell$ and the search points with exactly $n/2$ ones have a non-zero function value. Let us assume that we start in such a search point x with $\textsc{Jump}_\ell(x) = \text{OM}(x) = n/2$. Sampling the optimum directly from x seems hopeless, as all we could do is to set the mutation strength to $n/2$ and hope that exactly the n/2 zero-bits in x are chosen to be flipped. The probability of this event is $(\binom{n}{n/2})^{-1}$, i.e., negligibly small. We therefore need to approach the optimum if we want to find more efficient solution strategies. The main idea developed in [DDK15] is to find a strategy that allows to imitate the optimization of OneMax through RLS, i.e., by iteratively getting closer to the optimum. To this end, whenever we sample a new point with function value zero, we aim to understand if it is closer to the optimum as the previous best x or not. To learn this, we apply the following strategy: assume that the distance of x to the optimum is known to be d. When creating y from x by flipping exactly one bit, the distance of y to the optimum is either $d-1$ or $d+1$, and the distance to the "layer"

in the hypercube with non-zero function values is thus $n/2 - d + 1$ or $n/2 - d - 1$, respectively. To determine in which case we are, we sample sufficiently many strings at distance $n/2 - d - 1$, until we have obtained a reliable estimate for the average fraction of points at distance $n/2 - d - 1$ from y that are on this middle layer with fitness $n/2$. Since this fraction is different for points at distance $d - 1$ and $d + 1$ from the optimum, we can then infer in which case we are. With this strategy at hand, the main difficulty in determining the running time is in bounding by a polynomial expression the number of samples that are needed to distinguish the case $\mathrm{OM}(y) = n - d + 1$ from the case $\mathrm{OM}(y) = n - d - 1$. In order to minimize the number of samples required, we choose it *adaptively*, depending on the estimated number of ones in y. We also allow fairly frequent incorrect decisions, as long as the overall progress to the optimum is guaranteed.

From a high-level perspective, our proof can be seen as a confirmation that taking into account the (empirical) expected function values of the offspring can be a factor to consider in the replacement step of IOHs. This idea can indeed be found, explicitly or implicitly, in several evolutionary algorithms. Essentially, it can be seen as the backbone of so-called self-adaptive algorithms, where the search points "inherit" some values from their parents, and good values are more likely to survive because they generate "fitter" offspring (see Chapter 4 for a more detailed discussion of such parameter control mechanisms).

Coming back to the results in [DDK15], we prove the upper bounds in the following table, which summarizes the bounds for the k-ary unbiased black-box complexity of $\textsc{Jump}_\ell$, for different combinations of k and ℓ. The $\Omega(n \log n)$ lower bound for arity $k = 1$ follows from [LW12, Theorem 6], and the $\Omega(n)$ lower bound for the extreme jump function is again an easy application of Yao's minimax principle. For the latter, note that the we need to learn n bits of information, but each function evaluations reveals only a constant number of bits (since there are only three possible function values).[2]

Arity	Short Jump $\ell = O(n^{1/2-\varepsilon})$	Long Jump $\ell = (1/2 - \varepsilon)n$	Extreme Jump $\ell = n/2 - 1$
$k = 1$	$\Theta(n \log n)$	$O(n^2)$	$O(n^{9/2})$
$k = 2$	$O(n)$	$O(n \log n)$	$O(n \log n)$
$3 \leq k \leq \log n$	$O(n/k)$	$O(n/k)$	$\Theta(n)$

Theorem 2.4.5 (Table 1 in [DDK15])**.** *The k-ary unbiased black-box complexity of $\textsc{Jump}_\ell$ satisfies the bounds stated above.*

For long jump functions, we did not only show the bounds stated above, but we could reduce the k-ary black-box complexity to the $(k-2)$-ary unbiased black-box complexity of OneMax. The bounds in Theorem 2.4.5 then follow from our work [DW14c], with the exception of the 3-ary bound for the long jump functions, which required a separate proof in [DDK15].

Theorem 2.4.6 (Theorem 6 in [DDK15])**.** *Let $\ell \leq (1/2 - \varepsilon)n$. For all $k \geq 3$, the k-ary unbiased black-box complexity of $\textsc{Jump}_\ell$ is $O(UBBC_{k-2}(\textsc{OneMax}))$, where $UBBC_{k-2}$ denotes the $(k-2)$-ary unbiased black-box complexity.*

The strategies used to prove this theorem were later applied in [BDK16] to derive tight upper bounds for the unrestricted black-box complexities of the $\textsc{Jump}_\ell$ functions.

2.4.3 The Elitist Black-Box Model

A common trade-off that IOHs need to address is whether they rather *exploit* the knowledge about the problem instance at hand, or whether they use the samples to *explore* regions of the decision

[2] Strictly speaking, we do not need to "learn" in the unbiased model where the optimum is, but since unbiased algorithms treat all instances in the class $\mathcal{F}(\textsc{Jump}_\ell)$ identically, we can infer the argument from there. Note that the lower bound is not specific to the unbiased model, but applies to the unrestricted black-box complexity of $\mathcal{F}(\textsc{Jump}_\ell)$.

Algorithm 3 A $(\mu + \lambda)$ elitist black-box algorithm maximizing a function $f : \mathcal{S} \to \mathbb{R}$

1: Sample μ points $x^{(1)}, \dots, x^{(\mu)} \in \mathcal{S}$ and set $X \leftarrow \{x^{(1)}, \dots, x^{(\mu)}\}$
2: Query the ranking $\rho(X, f)$ of X induced by f
3: **while** termination criterion not met **do**
4: Depending on X and $\rho(X, f)$ choose a probability distribution $D^{(t)}$ on $\mathcal{S}^\lambda$
5: Sample $y^{(1)}, \dots, y^{(\lambda)} \in \mathcal{S}$ from $D^{(t)}$
6: Set $X \leftarrow X \cup \{y^{(1)}, \dots, y^{(\lambda)}\}$ and query the ranking $\rho(X, f)$ of X induced by f
7: **for** $i = 1, \dots, \lambda$ **do** Select $x \in \arg\min X$ and update $X \leftarrow X \setminus \{x\}$

space for which the uncertainty about the quality of the search points is high. This trade-off is very evident in surrogate-assisted optimization, where different infill criteria are used to maximize the information gain, the expected improvement, or the probability of improvement. Some evolutionary algorithms address this trade-off by striving to maintain some diversity in their "populations": they guide the decision which points to keep in the memory by a criterion that aims at maximizing the coverage of the decision space. However, it is also not uncommon to greedily keep only best-so-far solutions, in particular when the available memory ("population size") is small. Such selection rules are called *elitist selection,* and are often seen as an algorithmic interpretation of Darwin's "survival of the fittest" phenomenon. In particular for single-point algorithms, which always keep only one previously evaluated solution in memory, elitist selection is fairly common. In Section 2.4.2.1 we have seen that in combination with unary unbiased sampling, this strategy is at least almost optimal for optimizing the OneMax problem. However, it is also easily seen that elitist selection is very inefficient when optimizing, for example, the Trap function, which equals the OneMax problem but which changes the function value of the all-zeros string to $n+1$, making this string the unique global optimum of Trap. In this case, the expected optimization time of a unary unbiased elitist algorithm is easily seen to be exponential in n, since the algorithm cannot get closer to the optimum than its initial distance, which is $(1+o(1))n/2$ with overwhelming probability. A non-elitist single-point unary unbiased algorithm, however, can simply minimize the Trap function, which will guide the search towards the optimum in at most $O(n \log n)$ steps.

To formally analyze the impact of elitist selection on the running time of IOHs, we introduced in [DL17a] a family of $(\mu + \lambda)$ *elitist black-box models*. These models combine features of previous black-box models with an enforced *truncation selection,* as elitist selection is referred to in a less ambiguous way (see [DL17a] for different interpretations of the term "elitist selection"). The general structure of algorithms admitted in the elitist black-box models is provided in Algorithm 3. Different restrictions on the sampling distributions give rise to different models. For example, we may only want to consider unbiased distributions, or only distributions that depend on the *ranking* of the search points but not on their absolute function values. Note also that several – seemingly minor – decisions, such as the adaptive or non-adaptive initialization or the tie-breaking rule in the truncation selection, can have an important impact on the complexity of a best possible heuristic.

Apart from showing exponential performance gaps between the elitist and the non-elitist black-box models with various combinations of other restrictions, we also proved in [DL17a] that the strategies presented in Section 2.4.2.2 cannot be simulated efficiently.

Theorem 2.4.7 (Theorem 9 in [DL17a])**.** *For $\ell = 0$ the unary unbiased (1+1) elitist black-box complexity of the jump function Jump_ℓ is $\Theta(n \log n)$. For all $1 \le \ell \le n/2 - 1$ it is $\Theta(\binom{n}{\ell+1})$. In particular, for $\ell = \omega(1)$ the black-box complexity is superpolynomial in n and for $\ell = \Omega(n)$ it is $2^{\Omega(n)}$, in contrast to the situation described in Section 2.4.2.2.*

Fixed-probability black-box complexity: Another contribution of [DL17a] worth mentioning is an extension of the black-box complexity measure introduced in Equation (2.1) to a *fixed-probability*

performance measure.[3] Where classical black-box complexity is based on the *expected* running time of an algorithm, the *p-fixed-probability $\mathcal{A}$-restricted black-box complexity of a set $\mathcal{F}$ of problem instances* is defined as

$$\min\{T \mid \exists A \in \mathcal{A} \forall f \in \mathcal{F} : \Pr[T(A, f) \leq T] \geq p\}.$$

This notion was triggered in [DL17a] by the observation that restarts are not possible in the (strict interpretation of the) elitist black-box model, because the new starting point(s) would need to be at least as good as the previous-best solution(s). In practice, however, restarts are a common technique to handle situations in which an algorithm is "stuck" or does not show sufficient progress over a certain amount of time. In the black-box complexity literature, many results are based on algorithms that find the optimum in a certain number of steps with at least constant probability. By *randomly restarting* these algorithms after an appropriate number of steps, the expected running time can be bounded (see [DKLW13, Remark 7] for an explicit formulation). This is not possible in the elitist black-box model.

We will see applications of the fixed-probability black-box complexity measure in Section 2.4.3.1 and will also come back to it again in Section 2.5, in the outlook for this chapter.

Non-applicability of Yao's minimax principle: An important difficulty in the analysis of elitist black-box complexities is the fact that Yao's minimax principle [Yao77] cannot be directly applied to the elitist black-box model, since in this model the previously exploited fact that randomized algorithms are convex combinations of deterministic ones does not apply, see [DL17a, Section 2.2] for an illustrated discussion. Since Yao's minimax principle is *the* most important tool for proving lower bounds in the black-box complexity context, we need an efficient, yet powerful workaround. A natural strategy is to extend the collection $\mathcal{A}$ of elitist black-box algorithms to some superset $\mathcal{A}'$ in which every randomized algorithm *can* be expressed as a probability distribution over deterministic ones. Finding extensions $\mathcal{A}'$ that do not deteriorate the lower bounds (too much) is the main difficulty to overcome when applying this strategy, which is crucial for several bounds proven in [DL17a], but also for the main result in Section 2.4.3.2 below.

Additional challenge in bounding the information encoded by the state of an elitist black-box algorithm: One may be tempted to believe that, in the elitist model, one cannot learn information from search points that have strictly lower fitness than the current best solution, as such (seemingly inferior) search points have to be discarded immediately and can therefore not influence the sampling distribution of the next query. However, one has to be very careful with such arguments, as we illustrated in [DL18] with the following example: assume that there is a search point x from which we sample search point y_1 with some very small probability ε and we sample search point y_2 otherwise; i.e., we sample y_2 with probability $1 - \varepsilon$. For the sake of the argument assume further that for all search points $z \neq x$ the probability to sample y_1 or y_2 is zero. If, at some stage of the algorithm, we happen to have y_1 in the memory, we may then conclude that we must have been at x in the previous step. Moreover, if $f(y_2) > f(x)$ then with probability $1 - \varepsilon$ we would have proceeded to y_2, and thus we would never have visited y_1 (as we cannot return to x from a fitter search point). Therefore, by Bayes' theorem $f(y_2) \leq f(x)$ with probability at least $1 - \varepsilon$. Thus, although we have not visited y_2, we can deduce information about its fitness. This situation poses a difficult challenge for proving lower bounds in the elitist black-box model.

[3] Note that these measures are introduced in [DL17a] under the notion *p-Monte Carlo black-box complexity*. In the context of this thesis, however, and taking into account recent works on *fixed-budget* (see [JZ14] and follow-up works) and *fixed-target* ([BDDV20]) analyses, we prefer the term *fixed-probability black-box complexity*. We also invert here the meaning of p, which was the *failure probability* in [DL17a] and which is used here to denote the *minimal success probability*.

2.4.3.1 Results for OneMax

We have already mentioned in Section 2.4.1 the results on the combined memory-restricted ranking-based black-box complexity model. In fact, we also showed in [DL17b] that all the bounds proven for that model also hold in the elitist counterpart, i.e., with enforced truncation selection. However, unlike the results in Section 2.4.1, we can only show a p-fixed-probability black-box complexity in the elitist case. For the $(1+1)$ case, this results reads as follows.

Theorem 2.4.8 (Theorem 3 in [DL17b]). *For every constant $0 < p < 1$ there exists a (1+1) memory-restricted, ranking-based algorithm using truncation selection that with probability at least p finds the optimum of any* OneMax *instance using $O(n)$ function evaluations. This running time is asymptotically optimal.*

Similar results are shown for the extension of Theorem 2.4.2. Whether or not this result (and others proven in [DL17b]) extend to the classic black-box complexity (i.e., the one defined via expected optimization times) remains a challenging open problem, with limited practical, but probably significant methodological impact. In fact, as already discussed in [DL17b], a linear elitist black-box complexity for OneMax could be shown if the algorithm was granted only one additional bit that it could manipulate at will.

2.4.3.2 Results for LeadingOnes

In [DL18] we studied the (1+1) elitist black-box complexity of the LeadingOnes problem from Section 2.3.2, i.e., the generalization of LO to the class $\mathcal{F}(\mathrm{LO})$ of functions $\mathrm{LO}_{z,\sigma}$. Our main result is summarized in the following theorem.

Theorem 2.4.9 (Theorem 1 in [DL18]). *The (1+1) elitist black-box complexity of the class $\mathcal{F}(\mathrm{LO})$ is $\Theta(n^2)$. This bound also holds in the case that the algorithms can make use of the absolute fitness values of the search points in the population, and not only their rankings, i.e., in the (1+1) memory-restricted black-box model with enforced truncation selection.*

The $(1+1)$ elitist black-box complexity of LeadingOnes is thus considerably larger than its unrestricted one presented in Theorem 2.3.3, which we recall to be of order $n \log \log n$. Note also that the quadratic bound of Theorem 2.4.9 is matched by classical $(1+1)$-type algorithms such as the (1+1) EA, RLS, and others.

Our result is not the first quadratic lower bound for the LeadingOnes problem, since also its unary unbiased black-box complexity is of this order, as was shown by Lehre and Witt in [LW12]. Our result implies that even if we replace the mutation operator in RLS or in the (1+1) EA by a possibly strongly biased one, the resulting algorithm would still need time $\Omega(n^2)$ on average. This shows that not only unbiased sampling, but also the population structure of the algorithm and their selection strategies determine the comparatively slow convergence of these two well-known search heuristics.

A key step in the proof of Theorem 2.4.9 is to show that the amount of information that the algorithm has about the problem instance at hand is not sufficient to make substantial progress. What complicates this analysis is the fact that the LeadingOnes functions, in principle, allow for a rather important storage, since all the bits in the "tail" could – in principle – be used for encoding information. This is because all the bits in position $k+2, \dots, n$ can be changed without changing the function value of a search point with function value k and can hence be used to encode information. In other settings such as those described in Sections 2.4.1 or in [DW14a, DW14b], we have used such encoding techniques for storing information about previous samples, the number of iterations elapsed, etc. Here in the (1+1) elitist model, we need to show that this is not possible. Or, more precisely, we need to show that the information that an algorithm can store about the problem instance or the optimization trajectory cannot significantly speed up the search.

The intuitive reason why the given storage is not large enough is that, since the LeadingOnes problem is permutation-invariant, the black-box algorithms do not know *where* the irrelevant bits are located, and storing this information would require more bits than available. However, the discrepancy is rather small: in most parts of the process, if the number of bits of the storage space was larger by just a constant factor, then this would trivially allow for efficient use of the storage, and the lower bounds would break down. It is thus essential to find a good measure for the information that an algorithm can possibly encode in its queries. We develop a precise notion that bounds the amount of information that the algorithm has about the instance $\mathrm{LO}_{z,\sigma}$ at any given state. We can use this notion to estimate the gain that the algorithm can make, in terms of fitness increase, from any given amount of information. The main contribution of the work [DL18] is in making these intuitive concepts precise and utilizable in proofs.

In terms of future work, the proof of Theorem 2.4.9 suggests that the reason for the large complexity is rather the memory restriction than the selection strategy. We therefore conjectured in [DL18] that already the (1+1) memory-restricted black-box complexity of LeadingOnes is $\Omega(n^2)$. To date, this question is still open. Another interesting question posed in [DL18] is whether or not a memory-restricted black-box algorithm can benefit from trading-off a search point of better fitness value for one possibly containing more information (but having lower fitness). For LeadingOnes, we believe that this is not true; that is, we conjectured in [DL18] that, for every (1+1) black-box algorithm eventually accepting search points of smaller fitness, there exists an elitist one with the same or better expected running time.

The conference version [DL16b] of this work received a best paper award at the ACM Genetic and Evolutionary Computation Conference (GECCO) in 2016.

2.5 Outlook

All the results stated above hold for the number of function evaluations needed until an *optimal* target has been identified. They furthermore focus on *average* (over all independent runs of the algorithm) *worst-case* (with respect to the problem instances in the class $\mathcal{F}$) running time. Extending black-box complexity models to other performance metrics is much needed, to reflect the increased interest in the theory of IOHs community to understand not only worst-case expected optimization times, but also other performance metrics. We believe in particular the following measures to deserve more attention:

Beyond *expected* performance: As already argued in Section 2.4.3, there are situations in which the expected performance is not (or not the only) measure of interest. For example, an algorithm finding the optimal solution in few steps with 99% probability, but not finding it in the remaining 1% has infinite expected optimization time. In practice, it may nevertheless be preferable over one that has a finite, but very large expected optimization time. Such situations are much better reflected by the concept of p-fixed-probability black-box complexities, where setting $p = 1/2$, for example, leads to a black-box complexity notion with respect to worst-case median performance (as opposed to worst-case average performance). Several alternative notions could also be defined, e.g., measuring the mean of some given fraction (or inner quantiles) of runs or applying some penalization approach as is used in the *penalized average runtime* PAR-c scores.[4] However, apart from our own works [DL17a, DL17b] we are not aware of any works in this direction.

Beyond *worst-case* performance: In all black-box results so far, we studied the performance with respect to a worst-case instance. This is also *the* most common complexity measure used in classical

[4]The PAR-c is defined as the average first hitting time plus $(1-p_s)cB/p_s$, with p_s being the probability of a successful run, B being the maximal budget after which an algorithm is stopped, and c being the penalty factor of the PAR-c measure. PAR-c scores are common in the Machine Learning literature. In the empirical analysis of numerical black-box optimization algorithms, the PAR-1 score is studied under the notion of *expected running times* (ERT [HAB+16]).

Computer Science contexts. But just as discussed above in the context of running time distributions, this view is not necessarily suitable for each and every context. Alternatives of interest include average-case complexity or complexity measured with respect to some percentiles of the instances.

Beyond *optimization* time: Another way to extend existing notions would be to deviate from the focus on identifying an optimal solution, and to focus on first hitting times for other targets, expressed as absolute values or in terms of absolute or relative difference to the value of an optimal solution. Typically, these are expressed as approximation ratios, i.e., we wish to identify solutions that are at most $\varepsilon\%$ worse than an optimal solution. Where the value of an optimal solution is not known, one can focus on absolute target values.

Combinations of all the above are also of interest. That is, in the long run, we could even think of black-box complexity results with respect to measures that capture both the anytime performance and the failure probabilities. The area under the cumulative distribution function would be such a measure, which is already commonly used in standard benchmarking platforms for IOHs [HAR+20, DWY+18b], but which has not yet been studied in the black-box complexity context, nor in the running time analysis of IOHs.

A first step for all the above would be to extend the classical running time results for IOHs in the proposed way. Once this has been achieved, methods to bound the performance for *all* IOHs can be discussed.

We finally mention two other important directions for future work:
(1) **Beyond the discrete case:** The literature on black-box complexity has a very strong bias towards discrete optimization problems. However, similar notions can be defined for continuous and for mixed-integer problems. A few works in this direction exist, see [TG06, FT11] for examples.

(2) A rather technical, but methodologically highly interesting question is that of identifying ways to **bound from below the complexity of the k-ary unbiased black-box model for $k > 1$.** We currently do not have any non-trivial bounds in this context, and this not even for for simple functions such as OneMax or LeadingOnes. An interesting avenue towards such bounds could be in the study of combined black-box models, as suggested above in Section 2.4.3 and as later also investigated in [BB19b].

Chapter 3

From Black-Box Complexity Theory to Algorithm Design

In Sections 1.2 and 2.2 we have motivated theoretical research on sampling-based optimization heuristics and the study of black-box complexity models, respectively, with the hope to generate insight that can help us design more efficient algorithms. We will demonstrate in this section how such a process can look like in practice. Concretely, we will present two examples, in which a new algorithmic design has been heavily influenced by previous black-box complexity studies.

We start our discussion by recalling that for some black-box models we obtained black-box complexities that are matched by standard optimization heuristics such as RLS or the (1+1) EA. This is the case, for example, for the asymptotic, big-Oh complexity of OneMax in the unary unbiased black-box model: RLS and the (1+1) EA are unary unbiased black-box algorithms and their $\Theta(n \log n)$ expected running time matches the $\Theta(n \log n)$ unary unbiased black-box complexity of OneMax proven in [LW12]. Another example is the LeadingOnes problem in the (1+1) elitist black-box model with its $\Theta(n^2)$ black-box complexity, which is again matched by both RLS and the $(1 + 1)$ EA. For several other models and problems, however, or when regarding complexities that are more precise than just in the big-Oh asymptotic sense, we have observed black-box complexities that are smaller than the expected running times of typical IOHs. Two possible reasons for this discrepancy exist: either the respective black-box models do not capture very well the complexity of the problems for heuristic optimization approaches, or there are ways to improve classical heuristics by new design principles. In the restrictive models discussed in Sections 2.4 (see [Doe20b] for more examples), we have seen that there is some truth in the first possibility: several black-box complexities increase considerably when the class of admissible optimization heuristics is restricted to sub-classes of algorithms that share some common properties of classic optimization heuristics. In this section, however, we will show that there is also some truth in the second option, in the sense that the existing algorithms may be sub-optimal. Put differently, we show that the gaps between black-box complexity and known running times are not only due to "too generous" black-box models, but that there is indeed room for new algorithmic ideas. We also demonstrate how such ideas an be obtained from the black-box complexity studies themselves.

Precisely, we discuss in this chapter two examples, which both focus on the unbiased black-box complexity of OneMax. We first look at the binary case in Section 3.1, and we revisit the unary case in Section 3.2. In both settings we derive algorithms whose expected optimization times match the best known (in the binary case) and the best possible (in the unary case) bounds, up to lower order terms.

3.1 The $(1 + (\lambda, \lambda))$ GA – An $o(n \log n)$ Genetic Algorithm for OneMax

Our first example is taken from [DDE15], whose preceding conference version [DDE13] received a best paper award at the ACM Genetic and Evolutionary Computation Conference (GECCO) in 2013. We presented in this work a binary unbiased black-box algorithm which solves OneMax using $o(n \log n)$ function evaluations. As recalled above from Section 2.4.2, no unary unbiased algorithm can achieve such a running time. This result therefore answered a long-standing open question in evolutionary computation: whether or not crossover operators can help speed up the optimization of "smooth" problems. A number of results exist which show similar advantages of crossover, for example [JW02, FW04, Sud05, DT09, DHK12, Sud12, DJK+13, DFK+16, DFK+18], but in all of these works, non-standard problems or operators are regarded, the results hold only for uncommon parameter settings, or they require substantial additional mechanisms like diversity-preserving selection schemes. The latter also applies to our own work [CD18], where we showed a constant-factor speed-up over the unary unbiased black-box complexity of OneMax via a rather simple genetic algorithm; see Section 5.1 for more details. To our knowledge, Theorem 3.1.1 was the first example that proves advantages of crossover in a natural algorithmic setting for a simple hill climbing problem, and this not only in terms of constant factors, but also in big-Oh asymptotic terms.

It was previously known, by our own result proven in [DJK+11], that the binary unbiased black-box complexity of OneMax is at most linear. The algorithm used in [DJK+11], however, is very problem-specific and very heavily exploits the structure of the OneMax problem – in particular the fact that OneMax is fully separable (i.e., the decision variables do not interact with each other and can therefore be optimized sequentially). Our $(1 + (\lambda, \lambda))$ GA does not make use of such information. It uses fairly standard algorithmic components: standard bit mutation, uniform crossover, and truncation selection. The novelty is in how these operators are combined with each other. We summarize in this section the design process that has let to this structure.

Since the unary unbiased black-box complexity of OneMax is $\Omega(n \log n)$, it was clear for the development of the $(1 + (\lambda, \lambda))$ GA that an $o(n \log n)$ unbiased algorithm must be at least binary. This has led to the question how recombination can be used to learn from inferior search points. The following idea emerged. For illustration purposes, assume that we have identified a search point x of function value $\mathrm{OM}_z(x) = n - 1$. From the function value, we know that there exists exactly one bit that we need to flip in order to obtain the global optimum z. Since we want to be unbiased, the best mutation seems to be a random 1-bit flip. This has probability $1/n$ of returning the unique optimum z. If we did this until we identified z, the expected number of samples would be n, and even if we stored which bits have been flipped already, we would need $n/2$ samples on average.

Assume now that in the same situation we flip $\ell > 1$ bits of x. Then, with a probability that depends on ℓ, we have only flipped already optimized bits (i.e., we flipped bits in positions i for which $x_i = z_i$ to $1 - z_i$), thus resulting in an offspring of function value $n - 1 - \ell$. However, the probability that the position j in which x and z differ is among the ℓ positions is ℓ/n. If we repeat this experiment some λ times, independently of each other and always starting with x as "parent", then the probability that j has been flipped in at least one of the offspring is $1 - (1 - \ell/n)^\lambda$. For moderately large ℓ and λ this probability is sufficiently large for us to assume that among the λ offspring there is at least one offspring in which j has been flipped. Such an offspring has function value $n-1-(\ell-1)+1 = n-\ell+1$ instead of $n - \ell - 1$. Assume now that there is such an offspring x' among the λ independent samples crated from x. When we compare x' with x, they differ in exactly ℓ positions. In $\ell - 1$ of these, the entry of x equals that of z. Only in the j-th position the situation is reversed: $x'_j = z_j \neq x_j$. We would therefore like to identify this position j, and to incorporate the bit value x'_j into x.

So far we have only used mutation, which is a unary unbiased operation. At this point, we want to compare and merge two search points, which is one of the driving motivations behind *crossover*. Since

Algorithm 4 The $(1 + (\lambda, \lambda))$ GA maximizing a function $f : \{0,1\}^n \to \mathbb{R}$ with offspring population size λ, mutation rate p, and crossover bias c. Definitions of the mutation operator $\text{flip}_\ell(\cdot)$ and the crossover operator $\text{cross}_c(\cdot,\cdot)$ can be found in Section 1.3

Choose $x \in \{0,1\}^n$ uniformly at random (u.a.r.) ▷ initialization
while termination criterion not met **do**
  Sample ℓ from $\text{Bin}(n,p)$ ▷ random mutation strength
  for $i = 1, \dots, \lambda$ **do** $x^{(i)} \leftarrow \text{flip}_\ell(x)$ ▷ mutation phase
  Choose $x' \in \{x^{(1)}, \dots, x^{(\lambda)}\}$ with $f(x') = \max\{f(x^{(1)}), \dots, f(x^{(\lambda)})\}$ u.a.r. ▷ truncation selection
  for $i = 1, \dots, \lambda$ **do** $y^{(i)} \leftarrow \text{cross}_c(x, x')$ ▷ crossover phase
  Choose $y \in \{y^{(1)}, \dots, y^{(\lambda)}\}$ with $f(y) = \max\{f(y^{(1)}), \dots, f(y^{(\lambda)})\}$ u.a.r. ▷ truncation selection
  if $f(y) \geq f(x)$ **then** $x \leftarrow y$ ▷ replacement step

x clearly has more "good" bits than x', a uniform crossover, which takes for each position i its entry uniformly at random from any of its two parents, does not seem to be a good choice. We would like to add some bias to the decision making process, in favor of choosing the entries of x. This yields to a biased crossover, which takes for each position i its entry y_i from x' with some probability $c < 1/2$ and which takes the entry from x otherwise. The hope is to choose c in a way that in the end only good bits are chosen. Where x and x' are identical, there is nothing to worry about, as these positions are correct already (and, in general, we have no indication to flip the entry in this position). So we only need to look at those ℓ positions in which x and x' differ. The probability to make only good choices (i.e., to select $\ell - 1$ times the entry from x and only for the j-th position the entry from x') equals $c(1-c)^{\ell-1}$. This probability may not be very large, but when we do again λ independent trials, then the probability to have created z in at least one of the trials equals $1 - (1 - c(1-c)^{\ell-1})^\lambda$. For suitably chosen parameter values c, λ, and ℓ, this expression is sufficiently large to gain over the $O(n)$ strategies discussed above.

Putting everything together, the $(1+(\lambda,\lambda))$ GA does the following. In the *mutation step*, the algorithm creates λ offspring from x using the $\text{flip}_\ell(\cdot)$ mutation operator, for a mutation strength ℓ sampled from a binomial distribution $\text{Bin}(n,p)$ (the mutation rate p is a parameter of the algorithm). From these λ offspring, an offspring x' with largest function value among all offspring is selected, ties broken at random. In the *crossover phase,* the algorithm then creates again λ offspring, by recombining x and x' using the biased crossover $\text{cross}_c(\cdot,\cdot)$ described above. From these λ recombined offspring, the algorithm chooses one that has largest function vales (for OneMax, ties can again be broken at random, but for other problems it can be better to favor individuals that are different from x, see [DDE15, Section 4.3] for examples). This selected offspring y replaces x if it is at least as good, i.e., if $f(y) \geq f(x)$. Algorithm 4 summarizes the $(1 + (\lambda, \lambda))$ GA.

As mentioned above, the main result in [DDE15] was an $o(n \log n)$ bound for the expected running time of the $(1+(\lambda,\lambda))$ GA on OneMax. We later improved this original $O(n\sqrt{\log n})$ guarantee to the bound stated below. In [Doe16], B. Doerr showed that this improved bound is best possible among all static parameter settings.

Theorem 3.1.1 (from [DDE15, DD18, Doe16])**.** *The $(1 + (\lambda, \lambda))$ GA is a binary unbiased comparison-based black-box algorithm. For mutation rate $p = \lambda/n$, crossover bias $c = 1/\lambda$, and $\lambda = \Theta(\sqrt{\log(n) \log\log(n) / \log\log\log(n)})$, the expected optimization time of the $(1 + (\lambda, \lambda))$ GA on OneMax is $O(n\sqrt{\log(n) \log\log\log(n) / \log\log(n)})$. No static parameter choice of $\lambda \in [1..n]$, $k \in [0..n]$, and $p \in [0,1]$ can give a better expected running time.*

Note that by setting $p = \lambda/n$ and $c = 1/\lambda$, as suggested by Theorem 3.1.1, the three-dimensional configuration space of the $(1 + (\lambda, \lambda))$ GA is reduced to a one-dimensional one, which is easier to deal

with. In fact, we rarely see running time bounds that depend on two or more parameters, a topic that is likely to gain importance in the coming years, as we shall briefly discuss in Section 6.

Coming back to the design principles that have driven the development of the $(1 + (\lambda, \lambda))$ GA, we note that what is unusual about the $(1 + (\lambda, \lambda))$ GA is not the set of operators it uses, but the order in which they are applied. In fact, the key idea in the $(1 + (\lambda, \lambda))$ GA is to use the crossover operator *after* the mutation step, with the hope to "repair" the damages done by the mutation operator, while maintaining its beneficial decisions. This idea was largely inspired by our black-box study on Mastermind (which we have described in Section 2.3.1), where we observed that efficient algorithms do not necessarily strive to make progress, but should also be able to learn from inferior solutions. We then realized that many classical optimization heuristics are not very efficient in learning from such search. Instead, they either tend to ignore them entirely, or they use them only to create some diversity in the population. After the Mastermind study, this approach seemed quite wasteful, especially when the heuristic is close to a local optimum and when, inevitably, a good fraction of the sampled points are worse than the current-best solutions.

We will improve the result of Theorem 3.1.1 in Chapter 4 by showing that the expected running time of the $(1 + (\lambda, \lambda))$ GA can be reduced to linear when the parameters are chosen in a dynamic fashion. In this case, the performance matches that of the best known binary unbiased black-box algorithm from [DJK+11]. We see this extension as one of the most important lessons learned from the $(1 + (\lambda, \lambda))$ GA, since it has triggered a lot of follow-up works on dynamic parameter settings. In fact, the whole next chapter is centered around this topic. Parameter control is today one of the most actively researched topics in the theory of evolutionary computation community, and this, to a large extent, thanks to the $(1 + (\lambda, \lambda))$ GA.

Apart from these extensions from [DD18], the $(1+(\lambda, \lambda))$ GA has seen a significant number of follow-up works. For example, it is shown in [BD17] that the $(1+(\lambda, \lambda))$ GA can solve random 3-SAT instances in the planted solution model having at least logarithmic average degree efficiently, using $o(n \log n)$ function evaluations. This work was inspired by the empirically robust performance of the $(1 + (\lambda, \lambda))$ GA reported in [GP14]. Performance analyses for the $\textsc{Jump}_\ell$ functions discussed in Section 2.4.2.2 are available in [ADK20]. It was shown there that the $(1 + (\lambda, \lambda))$ GA, with the right parameter setting, outperforms classical IOHs. The complex parameter setting was addressed in [AD20] by blending the $(1 + (\lambda, \lambda))$ GA with the "fast", heavy-tailed mutation operator from [DLMN17] (see Section 1.3 for definitions). In particular, the $(1 + (\lambda, \lambda))$ GA was shown to yield close to optimal performance for jump sizes up to $\ell = n/4$. Hybridizing the $(1 + (\lambda, \lambda))$ GA with the fast mutation operator was also the focus of [ABD20], where it was shown that this variant achieves linear expected optimization time on OneMax as well as promising empirical performance on random 3-SAT instances. A proof for the $\Theta(n^2)$ expected running time of the $(1 + (\lambda, \lambda))$ GA on LeadingOnes is available in [ADK19]. The $(1 + (\lambda, \lambda))$ GA has recently been extended for use in permutation-based problems [BB20].

3.2 Randomized Local Search with Self-Adjusting Mutation Strengths

In [DDY16] we presented another algorithm which was largely inspired by our previous black-box studies. Concretely, we wanted to understand in this work whether it is possible that the unary unbiased algorithm RLS can automatically detect the mutation strengths that were shown to yield close to best-possible performance on OneMax. That is, we took a deeper look at the results presented in Section 2.4.2.1, where we have shown that the RLS variant which chooses in each iteration the drift-maximizing mutation strength (i.e., the mutation strength which maximizes the expected progress) is almost optimal. Since progress is something that the algorithm can actively deduce from the function values, there was hope that we could design mechanisms which would automatically calibrate the mutation strength. In [DDY16] we then showed how this is indeed possible.

The core idea was to view the selection of the mutation strength as a multi-armed bandit problem, in which we have a portfolio of possible mutation strengths $\{k_i \mid i = 1, \dots, r\}$ and we attribute to each possible value k_i a *confidence* c_i, which is our estimate for the progress that we expect to see when applying this mutation strength to the current-best solution.[1] With the confidences at hand, we could greedily select at each step the mutation strength k_i with the largest value $c_i \in \arg\max\{c_j \mid j = 1, \dots, r\}$. This, however, bares the risk that we would not realize that a seemingly good mutation strength may have become sub-optimal as the optimization has progressed. We therefore decide to use the greedy selection in a $1 - \varepsilon$ fraction of the iterations only, and to reserve the remaining ε iterations for randomly selected mutation strengths k_j. More precisely, we decide in each iteration whether to use a greedy selection (with probability $1 - \varepsilon$) or whether we use a random mutation strength (with probability ϵ). This way, we balance the exploitation of good values and the exploration of possibly better suited ones. In the Machine Learning literature, this strategy is known as an *ε-greedy reinforcement learning strategy*. Note here that the key difference of our problem to the classic multi-armed bandit problem is in the fact that the *rewards* (measured here in terms of fitness progress $\max\{0, f(y) - f(x)\}$, with x denoting the current-best solution and y the "offspring" obtained by applying the mutation operator $\text{flip}_k(\cdot)$ to x) change over time (our problem is hence a "multi-armed bandit problem with dynamic rewards").

We did not discuss yet how we assign confidences to the different mutation strengths. Here, we first recall that the progress that can be achieved with a given mutation strength k changes with increasing fitness $f(x)$ of the current-best solution x (more precisely, it monotonically decreases with increasing $f(x)$, but this is irrelevant for the design of the strategy). Information obtained from previous iterations must therefore be discounted as the search progresses, to be able to adjust the confidences to the current situation. We have chosen an exponential time-decay; that is, we discount at each step the information from previous iterations by a factor of $(1 - \delta)$. We call δ the *forgetting rate*. Since $(1 - \delta)^{1/\delta} = 1/e + o(1)$ for all $\delta = o(1)$, the reciprocal $1/\delta$ of the forgetting rate is (apart from constant factors) the information half-life. With this exponential decay, we can now define the confidence $c_i[t]$ of the mutation strength k_i in iteration t as

$$c_i[t] := \frac{\sum_{s=1}^{t} \mathbf{1}_{k(s)=k_i} (1-\delta)^{t-s} (f(x_s) - f(x_{s-1}))}{\sum_{s=1}^{t} \mathbf{1}_{k(s)=k_i} (1-\delta)^{t-s}}, \tag{3.1}$$

where $k(s)$ is the mutation strength used in the s-th iteration, and x_s denotes the current-best solution at the end of that iteration. While this equation may look complex at first sight, the confidences c_i can be updated after each iteration without keeping in memory the full search history. Instead, it suffices to keep one quantity $w_i[t] = \sum_{s=1}^{t} \mathbf{1}_{k(s)=k_i} (1-\varepsilon)^{t-s}$ for each mutation strength k_i; see [DDY16] for details.

The main result in [DDY16] is then a proof showing that, for suitably selected hyper-parameters ε and δ, the above-described RLS variant almost always uses the best possible mutation strength when run on OneMax. More precisely, we showed that in all but a $o(1)$ fraction of the iterations the selected parameter value achieves an expected progress that differs from the best possible one by at most some lower order term. Consequently, the algorithm has the same optimization time (apart from an $o(n)$ additive lower order term) and the same asymptotic 13% superiority in the fixed budget perspective as the fastest algorithm which can be obtained when using the same portfolio of mutation strengths.

Theorem 3.2.1 (Theorems 1 and 2 in [DDY16])**.** *Let $\mathbb{E}[T(k_{\max})]$ be the minimal expected running time that any randomized local search algorithm using mutation strength between 1 and $k_{\max}$ can*

[1] In fact, in the original work [DDY16] we were not aware of the analogy of our strategy to the multi-armed bandit setting, and the presentation does therefore not mention this relationship. In [DDY16] we used the term *velocity* instead of *confidence*, and referred to this variable as v_i. Note also that the definitions of δ and ε are swapped in this section, compared to [DDY16], to maintain the analogy with the ε-greedy strategies known from the Machine Learning literature.

achieve on OneMax. *The expected running time T of the ε-greedy RLS variant with hyper-parameters $\epsilon = n^{-0.01}$, $\delta = n^{-0.99}$, and the same portfolio of possible mutation strengths is $\mathbb{E}[T(k_{\max})] + o(n)$.*

In the fixed-budget perspective, the following holds. Let $x_\varepsilon^{(t)}$ be the best solution that the ε-greedy RLS variant with this parameter setting has identified within the first t iterations. Similarly, let $x_{\text{RLS}}^{(t)}$ be the best solution that the classic RLS using 1-bit flips only has found within the first t iterations. For any $t \geq 0.2675n$, the expected Hamming distance to the optimum z satisfies $\mathbb{E}[H(x_\varepsilon^{(t)}, z)] \leq (1 + o(1))\, 0.872\, \mathbb{E}[H(x_{\text{RLS}}^{(t)}, z)]$.

The hyper-parameters in this result were taken as an example where this algorithm shows a superior performance. As noted in [DDY16], the particular choice of these parameters is not very critical. Clearly, ε has to be $o(1/\log n)$ to ensure that at most $o(n)$ iterations are performed with a sub-optimal mutation strength. Likewise, δ has to be $\omega(1/n)$ to ensure that information learned $\Omega(n)$ iterations ago (and thus at a time when the confi could be substantially different) has no significant influence on the current decision.

In addition to this theoretical result, [DDY16] also presents empirical results for the LeadingOnes and the minimum spanning tree (MST) problem. These experimental works suggest that, for suitably chosen hyper-parameters ϵ, δ, and $k_{\max}$, the average optimization time of the ε-greedy RLS variant can be significantly smaller than that of the $(1 + 1)$ EA. It even outperforms, empirically, RLS on LeadingOnes, and the RLS variant that always flips either one or two random bits in the current-best solution on the MST problem.

From a high-level perspective, we have learned from our black-box complexity studies that a drift-maximizing choice of mutation strengths can be almost optimal. Since the expected progress is a quantity that can be guessed from the search trajectory (at least to a sufficient extent, as the result above demonstrates), we designed a heuristic to automatically choose suitable mutation strength *on the fly.* As we shall see in the next chapter, the question how to design such *parameter control* strategies has evolved into one of the most active research areas within the theory of IOHs.

Chapter 4

From Algorithm Design to Parameter Control

We have discussed in the previous section that a dynamic choice of the mutation strength of RLS is beneficial for solving the OneMax problem. In evolutionary computation, such dynamic parameter settings are studied under the term *parameter control.* Parameter control aims at benefiting from a non-static choice of the parameters, with the underlying idea that the flexibility in the parameter choice can be used to adjust algorithms' behavior to the current state of the optimization process. For example, it may be beneficial to use a decreasing mutation rate, so that the algorithm can converge from global exploration to local exploitation. Another advantage of parameter control is the on *on-the-fly* identification of suitable parameter values; i.e., even when the gain of non-static parameter values is not very important, we may face situations in which we do not know how to set the parameter values. If we do not have previous experience to learn from, nor time or resources to set up parameter tuning studies, parameter control offers a possibility to automatically identify suitable values during the optimization process itself. That is, parameter control serves the *identification* of good parameter settings on the one hand, and the *tracking* of good configurations as they evolve during the optimization process.

Parameter control has a long research history in the domain of evolutionary computation (see [KHE15, AM16, LLM07, EHM99] for surveys), but used to suffer from weak theoretical support. Previous results mostly focused on the analysis of good parameter *schedules*, which do not use feedback from the optimization process to guide the search, but which use time alone as trigger to update the parameter setting. Many works furthermore focus only on the *existence* of good schedules, without explicitly recommending a way to generate these (the last two comments apply, in particular, to results on the Simulated Annealing heuristic proposed in [KGV83]; see [HJJ03, vLA87] for details). A notable exception to this rule are the *one-fifth success rule* (see Section 4.1) and a few results for parameter choices that explicitly depend on the quality of a best-so-far-solution ("fitness-dependent" parameter control in the taxonomy that we will describe in Section 4.2). This unsatisfactory situation has changed dramatically in the last five years. Starting with our work [DD18] (conference paper appeared in 2015 [DD15]), we have seen significant improvement in our theoretical understanding of parameter control. In this chapter, I will highlight a few selected examples. A more exhaustive survey for results that appeared before early 2018 and covering also results that are not my own, is available in our book chapter [DD20].

4.1 Self-Adjusting Parameter Values for the $(1 + (\lambda, \lambda))$ GA

Our interest in dynamic parameter choices originated in the work [DDE15] discussed in Section 3.1. We first observed that, similarly to the situation described in Sections 2.4.2.1 and 3.2, the expected running time of the $(1 + (\lambda, \lambda))$ GA can be improved from the bound proven in Theorem 3.1.1 to linear when the parameters are chosen in dependence of the function value of a best-so-far solution.

Theorem 4.1.1 (Theorem 8 in [DDE15])**.** *With x denoting the search point held in the memory at the beginning of the iteration, the expected running time of the $(1 + (\lambda, \lambda))$ GA with offspring population size $\lambda = \lceil \sqrt{n/(n - \mathrm{OM}(x))} \rceil$, mutation rate $p = \lambda/n$, and crossover bias $c = 1/\lambda$ on* OneMax *is linear.*

We later showed (see Theorem 11 in [DD18]) that the linear bound in Theorem 4.1.1 is asymptotically optimal, in the sense that no (static or dynamic) parameter setting can achieve a sub-linear expected running time on OneMax. This is, essentially, a consequence of the fact that the algorithm uses restricted memory combined with comparison-based selection rules.

We also performed in [DDE15] an empirical analysis, in which we tested the already mentioned *one-fifth success rule* to control the parameters of the $(1 + (\lambda, \lambda))$ GA. The one-fifth success rule was independently discovered in [Rec73, Dev72, SS68]. It suggests to set the step size of an evolution strategy[1] in such a manner that 20% of the iterations lead to a fitness improvement. The idea behind this is that when the success rate is higher, then most likely the step size is too small and time is wasted on minor improvements; however, when the success rate is smaller, then time is wasted by waiting too long for an improvement. The value 1/5 was derived from theoretical considerations for the performance of the (1+1) evolution strategy on the *sphere* problem $f : \mathbb{R}^n \to \mathbb{R}, x \mapsto \sum_{i=1}^{n} x_i^2$. Rechenberg showed that a success rate of about 20% yields optimal expected gain for this problem (and also on another problem with a so-called inclined ridge, see [Rec73] for details). An indirect interpretation of the one-fifth success rule was offered in [KMH+04]. It suggests to multiply the current step size by a multiplicative factor $F > 1$ in case an iteration was successful, and to multiply it by $F^{-1/4}$ otherwise. Note that this rule guarantees a stable parameter value when, on average, one out of five iterations is successful.

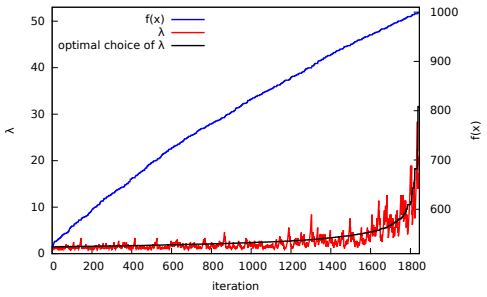


Figure 4.1: Evolution of the self-adjusting (red) vs. the optimal (black) choice of λ in one random run of the $(1 + (\lambda, \lambda))$ GA on OneMax, plotted against the iteration.

We applied this rule to the $(1 + (\lambda, \lambda))$ GA by setting $p = \lambda/n$ and $c = 1/\lambda$, and then controlling the value of λ. That is, we initialize λ by some value (we use $\lambda = 1$ in our experiments, but the initialization does not have a significant impact on performance, unless the initial values are unreasonably large). After each iteration, we replace λ by $F^{1/4}\lambda$ if the iteration was not successful in identifying a point of quality strictly better than the previous best, and we replace λ by λ/F otherwise. The empirical performance observed in [DDE15] seemed to match the linear running time already proven for the fitness-dependent rates from Theorem 4.1.1 very well. Figure 4.1 illustrates the evolution of the self-adjusting (red) vs. the optimal (black) choice of λ in one randomly selected run of the $(1 + (\lambda, \lambda))$ GA, plotted here against the iteration (which costs 2λ function evaluations each). For convenience, we also plot (in blue) the evolution of the quality of the best-found solution. The update strength F in this illustration is set to 1.5.

[1] Evolution strategies are the analog of evolutionary algorithms for the optimization of continuous problems $f : \mathbb{R}^d \to \mathbb{R}$. The step size is the analog of the mutation rate and determines the distribution of the distance at which the next solution candidates are sampled.

Motivated by the promising and stable performance of the self-adjusting $(1 + (\lambda, \lambda))$ GA, and using the drift analysis techniques that we have developed in previous works [DJW12], we could show that the one-fifth success rule achieves indeed asymptotically optimal expected running time on OneMax, provided the update strength F is not too large (see [DD18] for a discussion on how to chose F. Essentially, a value around or below 1.5 is recommended. An exhaustive evaluation of the hyper-parameter settings of various variants of the $(1 + (\lambda, \lambda))$ GA can be found in our empirical study [DD19]).

Theorem 4.1.2 (Theorems 9 and 11 in [DD18])**.** *For any sufficiently small update strength $F > 1$ the expected optimization time of the self-adjusting $(1 + (\lambda, \lambda))$ GA with mutation rate $p = \lambda/n$ and crossover bias $c = 1/\lambda$ has an expected linear running time on* OneMax*. This is asymptotically best possible among all parameter choices.*

No sublinear binary unbiased black-box algorithm is known for OneMax. The $(1 + (\lambda, \lambda))$ GA therefore matches the state-of-the-art performance of the highly problem-specific algorithm from our work [DJK+11].

As already mentioned in Section 3.1, Theorem 4.1.2 has triggered significant follow-up works. Apart from the works mentioned there, specific for the $(1+(\lambda, \lambda))$ GA, and the ones described in the remainder of this section, we wish to mention that the self-adjusting parameter control scheme introduced above has recently been improved in [BB19a] for more competitive performance on linear functions with random weights and for random satisfiable MAX-SAT instances.

4.2 Classification of Parameter Control Methods

The parameter control mechanism analyzed in the previous section uses a multiplicative update rule that depends on the success of a whole iteration. It therefore actively uses feedback from the optimization process, in contrast to the scheduled parameter updates used, for example, in Simulated Annealing. To categorize parameter control schemes according to their most prominent distinguishing factors, a taxonomy for parameter control was suggested in [EHM99]. Almost 20 years later, the focus of the evolutionary computation community has shifted significantly towards adaptive parameter control schemes, which motivated us to introduce a revised classification scheme in [DD20]. In our classification, we distinguish between the following control methods.

Class 1: State-Dependent Parameter Control. We classify as *state-dependent parameter control* those mechanisms that depend only on the current state of the search process, e.g., on the solutions currently held in the memory, their fitness values, their diversity, or on the time elapsed, the number of function evaluations or iterations performed. This category subsumes the "deterministic" category used in the classification scheme proposed in [EHM99] (which only contains time-dependent parameter choices) and it subsumes all other parameter setting mechanisms which determine the current parameter values via a pre-specified function that maps the algorithm states to parameter values or distributions over the portfolio of admissible values. All these mechanisms require the user to precisely specify how the parameter values or their probabilities shall depend on the current state. This, obviously, assumes a substantial understanding of the interaction between the algorithm at hand and the problem to be solved.

Class 2: Self-Adjusting Parameter Control. To overcome the usability challenges and the inflexibility of state-dependent parameter control mechanisms, several approaches to set the parameters in a *self-adjusting* manner have been proposed. Distinguishing features of self-adjusting parameter control mechanisms are (1) the use of *global parameters*, which are applied to all points in the population (as opposed to solution-specific ones used in the *endogenous* schemes described in Class 3) and (2) some sort of feedback from the optimization process. The two main subcategories of self-adjusting parameter schemes are:

- **Success-Based Parameter Control (1-step adaptation).** We classified in [DD18] as *success-based* parameter settings all those mechanisms which change the parameter value from one iteration to the next. That is, the values to be used in a current iteration depend only on the values at the beginning of the previous iteration and an assessment of how successful the last iteration was. The success measure can be a simple binary information like whether a solution with superior fitness was found, but it could also take into account quantitative information such as the fitness gain or loss in this iteration. Other statistics than the fitness can be taken into account, e.g., the evolution of the diversity of the population.

  The most common form of success-based rules are multiplicative updates of parameters, which increase or decrease the parameter value by suitable factors depending on whether or not the previous iteration was classified as success. The one-fifth success rule described in Section 4.1 is a classical example for such a multiplicative update. Success-based rules other than multiplicative updates have been designed as well. For example, in [DGWY19] the offspring were generated with two different parameter values and the parameter value which led to the best offspring determined the value of the next iteration.

- **Learning-Inspired Parameter Control (portfolio-based parameter selection).** The other main type of self-adjusting parameter control techniques are *learning-inspired parameter control mechanisms*. This class subsumes all those schemes which aim at exploiting a longer search history than just one iteration. To allow such learning mechanisms to also adapt quickly to changing environments, older information is taken into account to a lesser extend than more recent ones. This can be achieved by only regarding information from (static or sliding) *time windows* or by discounting the importance of older information via weights that decrease (usually exponentially) with the anciency of the data. Different from success-based parameter control, learning-inspired parameter control mechanisms maintain a portfolio of possible parameter values and try to learn which values are most suitable at a given point in time. The example of the ε-greedy RLS variant presented in Section 3.2 is a typical one for this category. Unfortunately, this is the only theoretical result available for this important parameter control scheme. Empirical examples can be found in [Thi05, CFSS08, FCSS08, FCSS10, LFKZ14]

Class 3: Endogenous Parameter Control (Self-adaptation). This category corresponds to the *self-adaptive* parameter control mechanisms in the taxonomy of [EHM99]. We prefer the name *endogenous* parameter control as it best emphasizes the structural difference of these mechanisms, which is to extend the original optimization problem by incorporating the parameter selection problem. This way, the parameter values become part of the search points, and undergo the same variation and selection process as the search points for the original optimization problem. The hope is that the points selected to remain in the memory of the algorithm also carry good parameter values. A simple way to incorporate the parameter selection problem is by attaching the chosen parameter values to the search points, see [Bäc92, SF96, Bäc98, KLR$^+$11] and [DWY18a, CL20, DL16a] for empirical examples and theoretical running time analyses, respectively.

In [DD20] we also proposed a fourth category, in which we collected so-called *hyper-heuristics.* However, as already noted in [DD20], this category was introduced mostly for historical reasons, which are not relevant in the context of this thesis.

4.3 One-Fifth Success Rule Revisited: Self-Adjusting Mutation Rates for the (1+1) EA

When we introduced the one-fifth success rule in Section 4.1, we explained that it was originally motivated by the adaptation of the step-size of an evolution strategy optimizing the continuous sphere

Algorithm 5 The $(1+1)$ EA$(p_{\text{init}}, p_{\min}, p_{\max}, A, b)$ maximizing a function $f : \{0,1\}^n \to \mathbb{R}$

1: Set $p = p_{\text{init}}$
2: Sample $x \in \{0,1\}^n$ uniformly at random and evaluate $f(x)$
3: **while** termination criterion not met **do**
4: $\quad y \leftarrow \text{SBM}(x, p)$
5: $\quad$ Evaluate $f(y)$
6: $\quad$ **if** $f(y) \geq f(x)$ **then** $x \leftarrow y$ and $p \leftarrow \min\{Ap, p_{\max}\}$ **else** $p \leftarrow \max\{bp, p_{\min}\}$

function. For other applications (and in particular for the discrete optimization problems considered in this thesis), there is no particular reason to believe that a success ratio of one to five should be optimal or anywhere close to being optimal. We empirically investigated this question in [DW18b] for the adaptation of the mutation rate of the $(1+1)$ EA on the two benchmark problems OneMax and LeadingOnes. This study then motivated a theoretical analysis for determining the optimal success ratio [DDL19]. We briefly summarize the main findings of these two works.

Empirical Analysis: We introduced in [DW18b] the $(1+1)$ EA$(p_{\text{init}}, p_{\min}, p_{\max}, A, b)$, which is summarized in Algorithm 5. The algorithm has five hyper-parameters: the initial mutation rate p_{init}, the minimal mutation rate $p_{\min}$, the maximal mutation rate $p_{\max}$, and the two update strengths $A > 1$ and $b < 1$. The algorithm uses standard bit mutation to generate one new solution per iteration. After each iteration, the algorithm updates the mutation rate: it replaces p by Ap whenever the new solution is at least as good as the previous best. In this case, the new solution also replaces the old one. It reduces p to bp when the new solution was worse than the previous best. The mutation rate is capped, if necessary, to remain in the boundaries $[p_{\min}, p_{\max}]$. The dependence on $p_{\min}$, $p_{\max}$, and p_{init} is very weak (see [DDL19] and [DW18a] for more details), and we therefore assume $p_{\min} = 1/n^2$, $p_{\max} = 1/2$, and $p_{\text{init}} = 1/n$ in the following and speak of the $(1+1)$ EA(A, b), for convenience. In [DW18b] we used the conditional standard bit mutation $\text{SBM}_{>0}$ introduced in Section 1.3 and we therefore refer to this algorithm as the $(1+1)$ EA$_{>0}(A, b)$.

We showed in [DW18b] that the (1+1) EA$_{>0}(A, b)$ yields very good performance on the two benchmark problems OneMax and LeadingOnes, and this performance is not only robust with respect to the hyper-parameters, but also with respect to the dimension n (if scaled by the asymptotic running times $n \log n$ and n^2, respectively). Figure 4.2 shows heatmaps with the empirical average running times for three selected scenarios. The values are averages over 101 independent runs, and the granularity of the data points is as follows: we vary A between 1.0 and 6.0 in multiples of 0.1 and we vary b between 0.00 and 1.00 in multiples of 0.02.

A visual inspection of the charts in Figure 4.2 shows that the performance landscapes for OneMax is quite flat; i.e., the bulk of the configurations achieves similar performance. This behavior, however, changes beyond some threshold values for A and b, as indicated by the quick succession of colors in the upper right corner. In this regime, small changes in the configuration can cause large changes in the performance. The heatmap for LeadingOnes is also quite flat. In the right-most plot of Figure 4.2 we zoom into the most interesting region of configurations with $A \leq 3$, $b \geq 0.4$, and show only average running times below 150 000. For comparison, the $(1+1)$ EA$_{>0}$ needs around 135 700 iterations, on average, on this problem instance, and RLS 125 000 iterations.

On LeadingOnes, around 67% of all 2,450 tested configurations with $A \in [1, 6]$ and $b \in [0, 1]$ have an average optimization time below $n^2/2$ – this is the expected optimization time of RLS. Between 37% ($n = 100$) and 41% (n=250) of all configurations are better than RLS by at least 10%. When restricting the configurations to those 450 that satisfy $1 < A \leq 2.5$ and $0.4 \leq b < 1$, around 78% of them are better than RLS, around 62% are better by at least 10%, around 30% outperform RLS by at least 15%, and still almost 20% of the configurations are better by at least 16%. From an algorithm

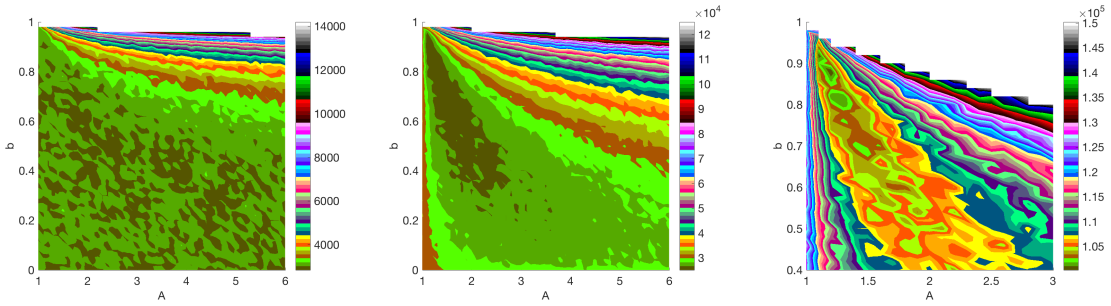


Figure 4.2: Optimization times of the $(1+1)$ $\text{EA}_{>0}(A,b)$ for different update strengths A and b, averaged over 101 independent runs: OneMax with $n = 500$ (left), LeadingOnes with $n = 250$ (middle), and LeadingOnes with $n = 500$ (right). Note that in the figure on the right we zoom into the interesting region with $A \leq 3$, $b \geq 0.4$, and average running time at most 150 000.

design point of view this is very good news: finding good hyper-parameters is not very difficult for this problem. We also observed that the numbers are very similar across all three tested dimensions $n = 100, 250, 500$, indicating that the good performance might translate to larger dimensions.

Theoretical Analyses: Motivated by the stable and promising results described in [DW18b], we analyzed in [DDL19] the performance of the $(1+1)$ $\text{EA}_{>0}(A,b)$ and of the $(1+1)$ $\text{EA}(A,b)$ by mathematical means. To keep the analogue to the one-fifth success rule, we set $A = F^s$ and $b = 1/F$ for some constant $F > 1$. This setting corresponds to a "$1/(s+1)$ success rule", since the mutation rate remains stable when one out of $s+1$ iterations was "successful" (where we recall that we consider a success if the new solution y replaces the old one x). We call s the *success ratio*.

The main result in [DDL19] shows that the $(1+1)$ $\text{EA}(A,b)$ the best performance is obtained for small update strengths $F = 1 + o(1)$ and success ratio $e - 1$ (i.e., a $1/e$ success rule).

Theorem 4.3.1 (Theorem 2.1 in [DDL19])**.** *The expected running time of the self-adjusting* $(1+1)$ *EA*(A,b) *with* $A = F^s$, $b = 1/F$, $F = 1 + o(1)$, *and constant success ratio* $s > 0$ *on* LeadingOnes *is at most* $\frac{s+1}{4\ln(s+1)}n^2 + o(n^2)$. *This expression is minimized for* $s = e - 1$, *with an expected running time of approximately* $0.68n^2 + o(n^2)$.

By a result proven in [BDN10], the $(1 + o(1))0.68n^2$ expected running time proven in Theorem 4.3.1 is asymptotically optimal among all $(1+1)$ EA variants that differ only in the choice of the mutation rates.

A key ingredient for proving Theorem 4.3.1 is a lemma which shows that, for suitably chosen hyper-parameters, the mutation rate used by the $(1+1)$ $\text{EA}(A,b)$ is, at all times during the optimization process, very close to the *target mutation rate* $\rho^*(\text{LO}(x), s)$, which we define as the unique mutation rate that leads to success probability $1/(s+1)$.

For the $(1+1)$ $\text{EA}_{>0}(A,b)$ variant analyzed in [DW18b], which uses the conditional mutation operator $\text{SBM}_{>0}$, we also proved a bound on the expected optimization time on LeadingOnes. We numerically evaluated this expression for problem dimensions up to $n = 10\,000$ and showed that the average running times coincide almost perfectly with the performance achieved by the best possible $(1+1)$ $\text{EA}_{>0}$ with optimally controlled mutation rates. More precisely, our numerical evaluation showed that the best running time is achieved for success ratio $s \approx 1.285$. With this choice (and using again $F = 1 + o(1)$), the performance of the self-adjusting $(1+1)$ $\text{EA}_{>0}(A,b)$ is almost indistinguishable from $(1+1)$ $\text{EA}_{>0,\text{opt}}$, the best possible $(1+1)$ $\text{EA}_{>0}$ variant using in each iteration the optimal mutation rate. Both algorithms achieve an expected running time for $n = 10\,000$ which is around $0.404n^2$.

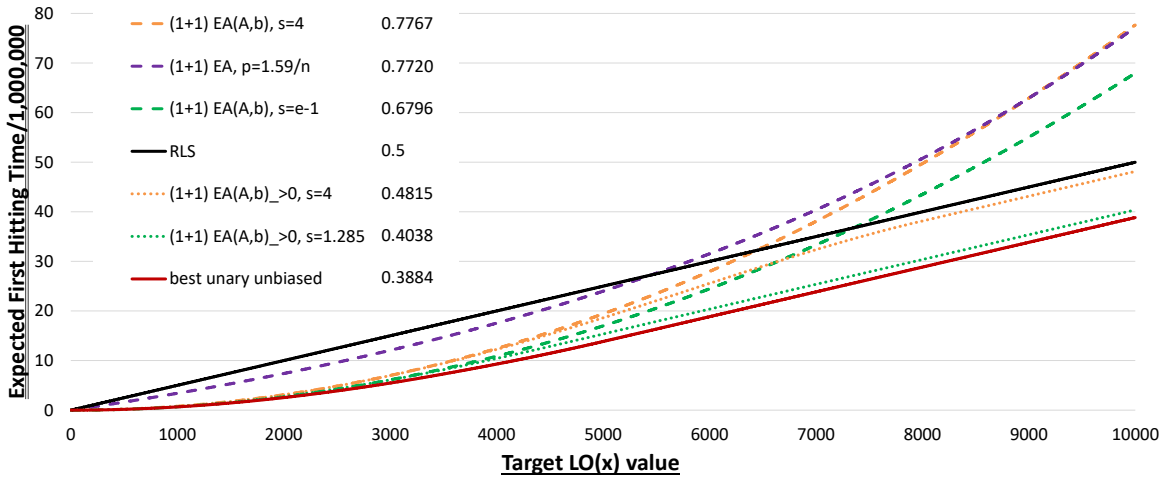


Figure 4.3: Expected fixed target running times for LeadingOnes in dimension $n = 10\,000$. The curve of $(1+1)$ $\mathrm{EA}_{>0,\mathrm{opt}}$ is indistinguishable from that of the $(1+1)$ $\mathrm{EA}_{>0}(A,b)$ with success ratio $s = 1.285$ and the curve of the $(1+1)$ $\mathrm{EA}_{\mathrm{opt}}$ indistinguishable from that of the $(1+1)$ $\mathrm{EA}(A,b)$ with success ratio $s = e-1$. The values shown in the legend are the expected optimization times, normalized by $1/n^2$.

For both algorithms, the $(1+1)$ $\mathrm{EA}(A,b)$ and the $(1+1)$ $\mathrm{EA}_{>0}(A,b)$, we did not only bound the expected optimization time but we also proved *stochastic domination bounds,* which provide much more information about the running time distribution and not only its first moment. The use of stochastic domination in running time analysis was motivated in [Doe19].

For LeadingOnes, it is also not difficult to extend optimization time results to fixed-target running times $T(v)$, which measure the number of function evaluations needed to find a solution x with solution quality $\mathrm{LO}(x) \geq v$. Figure 4.3 plots the expectation of these fixed target running times for some selected algorithms for $n = 10\,000$. We do not plot the $(1+1)$ $\mathrm{EA}_{>0,\mathrm{opt}}$, since its running time would be indistinguishable in this plot from the self-adjusting $(1+1)$ $\mathrm{EA}_{>0}$ with success ratio $s = 1.285$. For the same reason we do not plot $(1+1)$ $\mathrm{EA}_{\mathrm{opt}}$, the $(1+1)$ EA with optimal fitness-dependent mutation rate $p = n/(\ell+1)$, whose data is almost identical to that of the self-adjusting $(1+1)$ $\mathrm{EA}(A,b)$ with the optimal success ratio $s = e-1$. We also plot in Figure 4.3 the $(1+1)$ $\mathrm{EA}(A,b)$ with one-fifth success rule (i.e., success ratio $s = 4$). While its overall running time is the worst of all algorithms plotted in this figure, we see that its fixed-target running time is better than that for RLS for all targets up to 6 436. Its overall running time is very close to that of the $(1+1)$ EA with the best static mutation rate $p \approx 1.59/n$ [BDN10], and for all targets $v \leq 9\,017$ the expected running time is smaller.

The results presented in this section demonstrate the significant progress that running time analysis has seen in the last decade. Not only are we now able to analyze self-adjusting parameter choices, but we can even do so with very high precision.

4.4 Parameter Control for Multi-Valued Decision Variables

The main focus in the discussion above was on pseudo-Boolean optimization problems $f : \{0,1\}^n \to \mathbb{R}$. In [DDK18] we revisited the generalization of OneMax to larger alphabet sizes r, such as the Mastermind problem discussed in Section 2.3.1. More precisely, we studied the following three problems (where each *target string* $z \in [0..r-1]^n$ defines a new problem instance): $\mathrm{OM}_z^{(1)} : [0..r-1]^n \to$

$[0..n]; x \mapsto |\{i \in [1..n] \mid x_i = z_i\}|$ (this is the Mastermind problem with n positions and r colors), $\mathrm{OM}_z^{(2)} : [0..r-1]^n \to [0..n(r-1)]; x \mapsto \sum_{i=1}^n |x_i - z_i|$ (this is the L_1 version of Mastermind), and $\mathrm{OM}_z^{(3)} : [0..r-1]^n \to [0..n(r-1)]; x \mapsto \min\{x_i - (z_i - r), |x_i - z_i|, (z_i + r) - x_i\}$ (this is the L_1 version with ring topology on the decision space).

We designed in [DDK18] a multi-variate variant of RLS which uses the following self-adjusting parameter control scheme: For each coordinate i, we store a *velocity* v_i. This velocity can be seen as the individual mutation strength for position i. The RLS variant, $\mathrm{RLS}_{\mathrm{coord.}}(A, b)$, chooses in each iteration exactly one coordinate in which the entry is changed. When coordinate i is chosen, the current value x_i is replaced by $x_i - \lfloor v_i \rfloor$ or by $x_i + \lfloor v_i \rfloor$, with equal probability. As in the classical RLS algorithm, the so-generated string y replaces x if $f(y) \leq f(x)$ holds (assuming minimization of f as objective). The velocity v_i is updated as follows: if $f(y) < f(x)$ holds, v_i is replaced by Av_i (for some fixed constant $A > 1$), and it is replaced by bv_i otherwise, where $b < 1$ is again some fixed constant. The value of v_i is capped at 1 and at $\lfloor r/4 \rfloor$, respectively. The main result in [DDK18] is summarized in the following theorem.

Theorem 4.4.1 (Theorem 17 in [DDK18])**.** *For constants A, b satisfying $1 < A \leq 2$, $1/2 < b \leq 0.9$, $2Ab - b - A > 0$, $A + b > 2$, and $A^2 b > 1$, the expected running time of the $RLS_{coord.}(A, b)$ on any of the generalized r-valued* OneMax *function $\mathrm{OM}_z^{(i)}$, $i \in \{1, 2, 3\}$ and $z \in [0..r-1]^n$, is $\Theta(n(\log n + \log r))$. This is asymptotically best possible among all comparison-based variants of RLS and the $(1+1)$ EA. The running time is also by a multiplicative factor of at least $\log r$ better than any RLS or $(1+1)$ EA variant using* static *step sizes.*

In this theorem, the update strengths can be chosen, for example, according to a one-fifth success rule, by setting $A \in [1.6, 2]$ and $b = (1/A)^{1/4}$.

The result presented above is not only one more example which shows that we are now capable of analyzing parameter control mechanisms in various settings, but we also mention this work in this thesis because we believe that the topic of multi-variate optimization problems has not received the attention in the theory community that it should have. We hope to see more results for this important class of problems in the future.

Chapter 5

Selected Other Topics

After having summarized in the previous chapters the key results for two of my major research themes – black-box complexity and parameter control – we briefly discuss in the following sections a few selected results on different topics. For reasons of space we can only present the main results; readers interested in more detailed discussions and related works are referred to the original papers.

5.1 A Simple Proof for the Usefulness of Crossover

When we introduced in Section 1.3 some common mutation operators, we listed three different variants of standard bit mutation, SBM, $\text{SBM}_{>0}$, and $\text{SBM}_{0\to1}$. The latter two differ from the common version SBM only in how they treat mutation strength $k = 0$. Whereas SBM allows such mutation strengths, the operators $\text{SBM}_{>0}$ and $\text{SBM}_{0\to1}$ distribute the probability mass of sampling $k = 0$ to all other values proportionally or to $k = 1$, respectively. In many cases, these modifications will only affect constant factors of the expected running time. It is therefore often ignored in theoretical analyses, in particular when the main focus is on big-Oh asymptotic running times only. However, in practice one would like to avoid evaluating search points that have already been evaluated in previous iterations and it is therefore common practice to use $\text{SBM}_{>0}$ or $\text{SBM}_{0\to1}$ or yet different strategies which avoid mutation strengths $k = 0$. In [CD18] we demonstrated why the constant factor differences should not be neglected when aiming for results that are more precise than big-Oh accuracy.

Concretely, we analyzed in [CD18] the effects of these strategies on the expected running time of a greedy $(\mu + 1)$ GA, previously introduced in [Sud17]. Our main result (Theorem 1 and Corollary 1 in [CD18]) shows that for $\mu \geq 2$ the expected optimization time of the greedy $(\mu+1)$ $\text{GA}_{>0}$ with static mutation rate $p = 1/n$ on OneMax is at most $(1 + o(1))\frac{e-1}{2} n \ln(n) \approx 0.859 n \ln(n) + o(n \ln n)$ and for $p = 0.773581/n$ it is at most $(1 + o(1))0.851 n \ln(n)$. In particular, this shows that the so-configured $(\mu + 1)$ $\text{GA}_{>0}$ has an expected optimization time that is better than that of any unary unbiased algorithm, according to our discussion in Section 2.4.2.1. Our result is hence another important and "pure" example for the usefulness of crossover. While the gain over RLS is only a constant factor, the appeal of our result lies in the simplicity of its proof, which uses fairly standard mathematical techniques and which can therefore be taught in a basic algorithms lecture. An empirical evaluation demonstrates that the superiority of the greedy $(\mu+1)$ $\text{GA}_{>0}$ over the best unary unbiased black-box algorithm already holds for quite small problem instances.

5.2 Re-Optimization for Structurally Similar Problem Instances

In all the above, we have focused on *static* optimization problems, i.e., the optimization of a function f which does not change over time. We always assumed to have no prior knowledge about the problem at hand, and therefore initialized the search in randomly chosen solutions. In practice, however, we often face situations in which similar problem instances need to be solved again and again, e.g., computing an optimal delivery schedules for a postal operator, where the shipping volumes vary from day to day, but typically not too drastically from one day or one week to the next.

In such *re-optimization* situations, i.e., when we face a problem instance for which we know (or can guess) to have solved a similar instance in a previous task, we can hope to find good solutions by building on the knowledge derived from the previous one. A straightforward idea would be to simply initialize the search in the best solution that could be identified for the similar problem instance. In [DDN19] we proved that evolutionary algorithms do not necessarily benefit from such an approach. More precisely, we showed that the (1+1) EA can require $\Omega(n^2)$ function evaluations to re-optimize a LeadingOnes instance even if it is started in a Hamming neighbor of the optimum. Since this is asymptotically identical to the expected running time when starting from a random solution, there is no significant advantage from starting the search in a structurally good solution.

We then used the insights obtained from this negative result to suggest ways to use the "warmstarting" initialization in a better way. Concretely, the negative result for the (1+1) EA is caused by the fact that the good initial solution is quickly replaced by structurally worse solutions of equal or better fitness. Put differently, it is caused by the poor fitness-distance correlation of the LeadingOnes function. We can by-pass this bottleneck by maintaining a population of solutions around the initial search points. More precisely, we store for each distance level i the best-so-far solution $x^{(i)}$ at distance exactly i from the initial search point (which we recall was the good solution from the similar problem instance). In each iteration, we choose to center the search around the best-so-far search point (with probability 1/2) or we chose one of the $x^{(i)}$ otherwise (applying a uniform selection among all these $x^{(i)}$). This way, we prevent the search from drifting away too fast from the old, structurally good solution. We proved that our approach reduces the expected running time for above-mentioned LeadingOnes re-optimization problem to $O(\gamma\delta n)$, where δ is the Hamming distance of the new optimum to the starting point and γ is an estimated upper bound on this distance. The value of γ determines the largest distance i at which we store a point $x^{(i)}$.

We also showed in [DDN19] similar results for the optimization of linear functions with changing constraints and for the minimum spanning tree problem with added or deleted edges.

An interesting follow-up question arising from this work is how the above-sketched ideas can be used in situations in which the problem instance changes more frequently, and in particular before an optimal solution for the previous instance has been found.

5.3 One-Shot Optimization for Continuous Optimization

We now leave the world of discrete optimization problems and discuss in this section results for the optimization of continuous problems $f : \mathbb{R}^d \to \mathbb{R}$. We start with a theoretical result in Section 5.3.1 and then briefly describe two empirical results in Section 5.3.2. Both sections deal with *one-shot optimization*. We recall from Chapter 1 that one-shot optimization investigates situations in which we evaluate a fixed number of solution candidates $x^{(1)}, \dots, x^{(\lambda)}$ that have to be determined in a non-adaptive manner *prior* to the first evaluation.

5.3.1 Theoretical Result

A simple, yet effective strategy for one-shot optimization is to choose the λ candidates from a normal distribution $\mathcal{N}(\mu, \sigma^2)$ centered around an *a priori* estimate μ of the optimum and using a variance σ^2 that is calibrated according to the uncertainty with respect to the optimum. In [MDRT20] we studied the performance of this strategy when the optimum x^* is known to be distributed according to the standard Gaussian $\mathcal{N}(0, I_d)$ and the objective is to sample a point close to this optimum, in terms of Euclidean distance. Formulated as a classic optimization task, the problem corresponds to minimizing the sphere function $f_{x^*} : \mathbb{R}^d \to \mathbb{R}, x \mapsto ||x - x^*||^2$.

A previous result published in [CCD+20] showed that unless the sample size λ grows exponentially in the dimension d, the median quality of sampling from the prior distribution $\mathcal{N}(0, I_d)$ is worse than that of sampling a single point: the center $(0, \ldots, 0)$. This raises the question how to optimally scale the variance σ^2, a question that we answered in [MDRT20] in the following way: we derived sufficient and necessary conditions on the scaling factor σ such that, for $x^{(1)}, \ldots, x^{(\lambda)}$ sampled from $\mathcal{N}(0, \sigma^2)$,

$$\mathbb{P}\left[\min_{1 \leq i \leq \lambda} \|x^{(i)} - x^*\|^2 \leq (1 - \epsilon)\|x^*\|^2\right] \geq \delta,$$

for some $\delta \geq 1/2$ and $\epsilon > 0$ as large as possible. Concretely, we showed that setting $\sigma^2 = \min\{1, \Theta(\log(\lambda)/d)\}$ is asymptotically optimal, as long as λ is sub-exponential, but growing in d. Our variance scaling factor reduces the median approximation error by a $1-\epsilon$ factor, with $\epsilon = \Theta(\log(\lambda)/d)$. We also proved that no constant variance nor any variance which scales as $\omega(\log(\lambda)/d)$ can achieve such an approximation error.

We complemented our theoretical analyses by an empirical investigation of the rescaled sampling strategy. Experiments on the sphere function confirm the superiority of our approach. We also showed that the rescaled sampling yields good performance on two other benchmark problems, the Cigar and the Rastrigin function. Finally, we demonstrated that these improvements are not restricted to the one-shot setting but can also yield performance gains in the initialization of IOHs. More precisely, we show a positive impact on the initialization of Bayesian optimization algorithms [JSW98] and on differential evolution [SP97].

5.3.2 Empirical Works

Analyzing the impact of good one-shot optimization techniques on the initialization of surrogate-assisted optimization techniques was also the focus of our work [BDK20a]. We considered the efficient global optimization algorithm (EGO [JSW98]) and compared four different initialization strategies, which sample the initial set of points via uniform sampling, Latin Hypercube Sampling (we used the "improved" LHS design as suggested in [BG02]), *Sobol' sequences* [Sob67], and *Halton designs* [Hal60], respectively. We also analyzed the impact of the *size* of the initial point set, expressed as a percentage of the total budget. Concretely, we consider six different total budgets $n \in N := \{2^x \mid x \in \{4, \ldots, 9\}\}$, and ten different initial design sizes $\lambda = \lceil k \cdot n \rceil$ with $k \in K := \{0.1, 0.2, \ldots, 1.0\}$. Our test bed are the 24 benchmark functions from the BBOB collection [HFRA09] of the COCO environment [HAR+20]. We perform our experiments in five different dimensions $d \in D := \{2, 3, 4, 5, 10\}$.

Overall, we found that small initial budgets using Halton sampling seem to be preferable over large budgets for initialization and/or other sampling techniques. However, the performance landscape is rather unstructured, and further investigations are needed to derive decision rules which sampling method to prefer in which cases. What concerns the size of the initial point set, our results suggest an adaptive choice. This is easy to implement in surrogate-assisted optimization techniques, and can significantly improve performance, as we have demonstrated in our work. In fact, we even identified

extreme cases in which using the total budget for non-adaptive sampling (i.e., setting $k = 1$ in the notation introduced above) outperformed any other choice of k. This also makes the connection to one-shot optimization, the topic of this section. Our findings strongly suggest to extend our investigations of the power and limits of adaptive sampling (which we have started for the discrete case by the results presented in 2.3) to the continuous case. Surrogate-assisted optimization offers an ideal setting for this research question.

Note that the Halton point sets, which gave the best overall results in [BDK20a], are so-called *low-discrepancy* point sets. The discrepancy is a measure for the regularity of distributions, see [DP10, Mat09, Nie92] for surveys of this important measure from Numerical Analysis.[1] Low-discrepancy point sets have been shown to perform well in one-shot optimization tasks, see [CCD+20] for the current state of the art and further references. However, despite several favorable comparisons of low-discrepancy point sets over random sampling or LHS designs, the question if discrepancy correlates well with good performance in one-shot optimization has not been investigated. We addressed this question in our work [BDK+20b]. Using again the 24 benchmark functions from the BBOB collection as test bed, we found that a positive correlation between discrepancy and one-shot performance can indeed be observed on the global scale, i.e., when aggregating over all the use cases. The precise relationship between functions and best samples, however, is again fuzzy, and deserves further investigation. Since one-shot optimization does not allow adaptation, a relationship between high-level features of the search space (such as its dimension, its (guessed) multimodality, separability, etc.) seems worthwhile to study.

Apart from one-shot optimization in the classical sense, where the aim is to minimize the simple regret $\min_{i=1,\dots,\lambda} f(x^{(i)}) - \inf f$, we also studied in [BDK+20b] *one-shot regression* (also studied under the term *global surrogate modeling*), where the objective is to build a suitable approximation $\hat{f}$ of the true objective function f. Here we made the interesting observation that some of 24 specifically designed point sets (one per each BBOB function) yield surprisingly good approximations $\hat{f}$ across the whole benchmark set, i.e., when evaluated on the other 23 BBOB functions that they have not been optimized for. These results give strong indication that significant performance gains over state-of-the-art one-shot sampling techniques are possible and that IOHs can be an efficient means to evolve these (because this is how we obtained the 24 sets).

5.4 IOHprofiler - A Versatile Benchmarking Environment

The vast majority of results described in this thesis are of theoretical nature; i.e., they derive some properties about the behavior of IOHs using a mathematical approach. As discussed in Section 1.2, we should nevertheless not forget that such mathematical analyses can only offer one side of algorithm analysis, often focused on structurally simple benchmark problems and algorithms. We also have to admit that the mathematical language in which we typically express our findings does not necessarily appeal to everyone, and may in particular not appeal to practitioners of IOH methods, who are often not trained to interpret asymptotic running time statements nor to read and interpret mathematical proofs. For a broader and more complete insight into the working principles of IOHs, but also for communication purposes, it is therefore desirable to complement our mathematical analyses by empirical evaluations.

Supporting the empirical analysis and comparison of algorithms through a systematic experimental design is one of the primary goals of *algorithm benchmarking*. Algorithm benchmarking addresses

[1] More precisely, several discrepancy measures exist. For the context of this thesis, we focus on the L_∞ *star discrepancy* measure, which for a given set X measures the largest deviation between the volume of an anchored box $([0, y_i])_{i=1}^d$ and the fraction $|\{x \in X \mid \forall i \in [1..d] : x_i \in [0, y_i]\}|/n$ of points that fall inside this box. Some of my works are centered around the star discrepancy measure: computational aspects are considered in [GWW12, GSW09, DGW14], the generation of low discrepancy point sets using genetic algorithms in [DR13], and in [DDG18] we derived a probabilistic lower bound for the star discrepancy of LHS designs. They are not further described here for reasons of space and scope.

the selection of problem instances that are most suitable for an accurate performance extrapolation, the experimental setup of the data generation, the choice of the performance indicators and their visualizations, the choice of the statistics used to compare two or more algorithms, etc. Unfortunately, algorithm benchmarking can be a rather repetitive and tedious task, especially when it comes to collecting, comparing, and interpreting performance data.

With the goal to mitigate the time and knowledge investment needed to carry out sound benchmark studies, the team around Thomas Bäck at Leiden University and I have developed a new benchmarking environment, which we call IOHprofiler [DWY+18b]. IOHprofiler is designed with a fine-grained, dynamic, and multi-faceted performance analysis of IOHs in mind. The tool therefore does not only track performance data, but records the evolution of control parameters, making IOHprofiler particularly useful for the analysis, the comparison, and the design of algorithms with dynamic parameters.

IOHprofiler has a modular design, built to integrate various elements of the entire benchmarking pipeline, ranging from problem (instance) generators and modular algorithm frameworks over automated algorithm configuration techniques and feature extraction methods to the actual experimentation, data analysis, and visualization. Notably, IOHprofiler already provides the following components:

- **IOHproblems:** a collection of benchmark problems. This component currently comprises (1) the PBO suite of pseudo-Boolean optimization problems suggested in [DYH+20], (2) the 24 numerical, noise-free BBOB functions from the COCO platform [HAR+20], and (3) the Wmodel problem generator proposed in [WW18].
- **IOHalgorithms:** a collection of IOHs. For the moment, the algorithms used for the benchmark study presented in [DYH+20] are included, but extensions for both combinatorial and numerical solvers are in progress. Notably, we are working on an extension towards the object-oriented algorithm framework ParadisEO [CMT04], as well as towards the modular algorithm frameworks for CMA-ES variants from [vRWvLB16] and for the hybridization between particle swarm optimization (PSO) and differential evolution (DE) proposed in [BWB20].
- **IOHdata:** a data repository for benchmark data. This repository currently comprises the data from the experiments performed in [DYH+20], some selected data sets from the COCO repository [HAB20], as well as performance data from Facebook's Nevergrad benchmarking environment [RT18], which can be fetched from their repository upon request.
- **IOHexperimenter:** the experimentation environment that executes IOHs on **IOHproblems** or external problems and automatically takes care of logging the experimental data. It allows for tracking the internal parameter of IOHs and supports various logging options to specify the granularity at which the performance and parameter data is recorded.
- **IOHanalyzer:** the module which provides detailed statistics about fixed-target running times and about fixed-budget performance of the benchmarked algorithms on real-valued, single-objective optimization tasks. Performance aggregation over several benchmark problems is possible, for example in the form of empirical cumulative distribution functions. Key advantages of IOHanalyzer over other performance analysis packages are its highly interactive design, which allows users to specify the performance measures, ranges, and granularity that are most useful for their experiments, and the possibility to analyze not only performance traces, but also the evolution of dynamic state parameters. Our key design principles are 1) an easy-to-use software interface, 2) interactive performance analysis, and 3) convenient export of reports and illustrations. The stable release of IOHanalyzer is available on CRAN.

IOHprofiler is fully open source. It is available on GitHub at [DWY+18c].

Our first use case of IOHprofiler was the comparison of different variants of the $(1+\lambda)$ EA on OneMax and LeadingOnes, which we presented in [DYvR+18]. Notably, the results inspired us to improve

the precision of the running time of the standard $(1 + \lambda)$ EA (i.e., the version using standard bit mutation with a static mutation rate of $1/n$) and for the $(1+\lambda)$ $\text{EA}_{>0}$ version, which uses the $\text{SBM}_{>0}$ mutation operator instead.

Theorem 5.4.1 (Theorem 1 in [DYvR+18]). *For all $n, \lambda \in \mathbb{N}$ the expected optimization time of the $(1 + \lambda)$ EA with static mutation rate $0 < p < 1$ on the n-dimensional* LeadingOnes *function is at most $1 + \frac{\lambda}{2} \sum_{j=0}^{n-1} \frac{1}{1-(1-p(1-p)^j)^\lambda}$ and the expected optimization time of the $(1 + \lambda)$ $EA_{>0}$ is at most $1 + \frac{(1-(1-p)^n)\lambda}{2} \sum_{j=0}^{n-1} \frac{1}{1-(1-p(1-p)^j)^\lambda}$.*

For $\lambda = 1$ both bounds are tight, by results proven in [BDN10, JZ11], respectively. For reasonable (i.e., not too large) values of λ the bounds in Theorem 5.4.1 are close to tight, see [DYvR+18] for a detailed discussion.

Extending the empirical results presented in [DYvR+18] for OneMax, we observed in [RABD19] that none of the algorithms studied in [DYvR+18] achieves stable performance for varying offspring population sizes λ. That is, while the ranking of algorithms remains identical for all tested problem dimensions n (we investigate up to $n = 100\,000$), the ranking is heavily influenced by the choice of λ. This motivated us to suggest a new parameter control technique in [BDR20], which addresses this influence. In a nutshell, our new control mechanism is a hybrid between a reinforcement learning strategy and a success-based multiplicative update rule (as discussed in Section 4.3, the latter can be seen as a generalized one-fifth success rule). The hybrid strategy shows stable performance on OneMax, for all tested offspring population sizes $\lambda \in \{2^i \mid i \in [0..12]\}$. For problems different from OneMax, however, the choice of the minimal "cut-off" mutation rate $p_{\min}$ as well as the dependence of the update rule on strict improvement or on a successful replacement of the center of search (the "parent") can have a decisive influence.

Building on the work [DYvR+18], a number of improvements were subsequently made to IOHprofiler, and the first study of an important number experiments was reported in [DYH+20], where we evaluated twelve benchmark algorithms on 23 different problems in four dimensions. In the meantime, IOHprofiler has been used in a number of studies, including [HBS19, YDB19, CSC+19, DD19, VWDB20, VWBD20, LM20, LGW+20].

In ongoing and future work, we plan to expand the compatibility of IOHprofiler with respect to automated algorithm configuration and design tools (in particular, we are currently working on interfaces with irace [LDC+16] and ParadisEO [CMT04], but an important step will also be the selection and integration of feature selection methods which can guide the automated design on new problems or problem instances [KHNT19, KT19, MSKH15, JD20]). We also plan on exploiting the fine-grained performance logs to develop algorithms that show good anytime performance. Finally, we will continue our efforts to collect scalable and meaningful benchmark problems, and to make them available through IOHproblems or through interfaces with other benchmarking platforms and problem collections.

Chapter 6

Outlook

We have summarized in this thesis some of my research activities from the last nine years. Reflecting the time spend on the different topics, we have mostly focused on two of them, black-box complexity and parameter control. In the last section we have then summarized some selected activities from other research themes. We now present what we consider to be general trends in the theory of sampling-based optimization heuristics, and where we expect the field to develop.

A re-occurring topic in this thesis were **high-precision analyses,** both for the complexity models as well as for the performance analysis of IOHs. A key driver of this development are so-called *drift theorems,* which allow us to translate bounds on the step-wise progress into performance guarantees for the whole optimization process. Drift theory has in the last decade developed into the single-most used method in the analysis of IOHs. See [Len20] for a recent survey of existing results and [MA19, AAG18] for applications of drift analysis in continuous optimization. We are convinced that drift analysis will continue to play an important role in the analysis of IOHs also in the next decade. By refining the conditions under which drift statements can be made, and by gaining in knowledge how to build suitable potential functions (which measure the progress or the "state" of the optimization process), we will extend the scope of problems and algorithms for which we can derive performance guarantees. Not less important, we will continue to see significant gains in the precision of these guarantees, and this with respect to

- the **concentration of the running times** (or complexity statements), and stochastic domination bounds as suggested in [Doe19],
- the **dependency of the performance guarantees on the control parameters** of the algorithms, and in particular with respect to more than one control parameter. First steps in this direction have been made in [GW17], where bounds for the expected running time of the $(1 + \lambda)$ EA are given which depend on both the offspring population size λ and the mutation rate p, and by the lower bounds that we have proven for the $(1 + (\lambda, \lambda))$ GA in [DD18].
- the **scope of the performance guarantees:** extending optimization time, we will see an increased interest in anytime measures, in terms of fixed-target, fixed-budget, and fixed-probability statements, but also in terms of aggregated anytime performance measures such as the cumulative distribution functions (CDF), which are standard today, thanks to the COCO benchmarking environment [HAR+20], in the empirical analysis and comparison of IOHs.

Related to the precision of performance guarantees is the quest for statements "**beyond the worst-case**"; that is, results that do not over-emphasize rare pathological cases, but give a realistic estimate

for a "typical" or a "likely" performance. While for most situations studied in this thesis, the worst-case assumption did not have any or no significant influence on the performance guarantees (since the algorithms would treat the instances more or less identically), the question how to aggregate performances gains importance when trying to measure, for example, the discrepancy between deterministic and randomized IOHs.

Fueled by applications in Machine Learning, we expect to see significant progress on sampling-based approaches for **mixed-integer optimization problems.** In this context, the theory field has much to catch up on, given that the by far dominant fraction of theoretical analyses of sampling-based optimization heuristics assumes *either* discrete *or* continuous decision spaces. Luckily, we see in the example of drift theory that the two communities are converging in terms of analytical methods, so that we can be optimistic to see running time analyses for mixed-integer optimization problems in the not-so-far future. Interesting use cases could be hyper-parameter optimization, since this is related to the parameter tuning and control questions already studied in the theory of IOHs. First progress on this topic has already been made, see [KLLG19, HOS20] and references mentioned therein.

What concerns parameter control, two important topics are (1) the **control of two or more parameters** and (2) efficient control strategies for **multimodal optimization problems,** i.e., problems with several local optima.

Another topic for which we hope to see progress in the coming years is the theoretical analysis of IOHs which are explicitly not unbiased, but **which actively learn influence and correlations between variables** to guide the search, such as the algorithms suggested in [TB20, BMAB20]. Results for such IOHs would bring the theoretical analyses closer to the algorithms used in practice.

Last but not least, we firmly believe that a **reinforced communication between theoreticians and practitioners** of sampling-based optimization methods is much needed, to raise awareness for the knowledge gained through mathematical and empirical analyses, respectively. As stated in Section 5.4, we believe that algorithm benchmarking can offer a suitable bridge between the two sub-communities. In particular, we see the following advantages that theoreticians can expect from using empirical benchmarking (a more exhaustive list can be found in Section 2 of our survey [BDB$^+$20]):

(1) Science communication and educational purposes: Benchmarking offers a convenient way to illustrate key insights by means that practitioners and researcher with a non-mathematical background are much more used to than the "theorem-proof" style of documentation common in theoretical papers. We are therefore convinced that benchmarking is an essential tool for a better adoption of insights that the theory of IOHs literature has to offer.
(2) Testing generalization: As we discussed in Section 1.2, theoretical results for sampling-based optimization heuristics are often restricted to problems and/or algorithms that are not representative of the complexity encountered in practice. Benchmarking offers a convenient way to test how far proven performance guarantees or algorithmic behavior extends to other problems. It also allows to obtain precise running time estimates, as opposed to the asymptotic results typically sought for by the mathematical approaches.
(3) Generating New Hypothesis: Benchmarking does not only serve demonstration purposes, but can also be a source for inspiration. We have seen examples in Sections 4.1 and 4.3, where the performance guarantees proven for the self-adjusting $(1 + (\lambda, \lambda))$ GA and the generalized one-fifth rule, respectively, were largely inspired by previous empirical studies from [DDE15] and [DW18b], respectively.

Curriculum Vitæ

Personal Data



Name:	Doerr (formerly Winzen), Carola
Date of birth:	March 05, 1984, in Würselen, Germany
Nationality:	German
Family status:	Married, two kids (born 04/2013 and 09/2015)
Homepage:	`http://www-ia.lip6.fr/~doerr/`
Publications:	Complete list, Google Scholar profile, DBLP entry

Education

01/10–12/11	Ph.D. studies in Computer Science (Dr.-Ing., with distinction, summa cum laude) Max Planck Institute for Informatics and Saarland University, Germany Supervisor: Prof. Dr. Dr. hc. mult. Kurt Melhorn
10/03–08/07	Studies in Mathematics (Dipl.-Math., "very good", best possible grade) Kiel University, Germany
07/03	Abitur (best possible grade of 1.0, two awards), Konstanz, Germany
08/01–07/02	AFS Intercultural Programs High-School Exchange in Tobatí, Paraguay

Current and Previous Academic and Industrial Positions

since 10/13	Permanent researcher with the French National Center for Scientific Research (CNRS), affiliated with the Computer Science department (LIP6) of Sorbonne Université, Paris, France
10/12–09/13	PostDoc at LIAFA (now IRIF), Paris Diderot University, France (funded by the Alexander von Humboldt Foundation)
01/12–09/13	PostDoc at Max Planck Institute for Informatics, Germany (part-time after 10/12)
01/10–12/11	PhD student at Max Planck Institute for Informatics, Germany
12/07–01/12	Business Consultant with McKinsey & Co., Munich office, Germany (on educational leave from 01/10, working part-time 10/11-01/12)
07/06–10/06	Internship with Deutsche Lufthansa AG, Frankfurt, Germany

Awards, Distinctions, and Fellowships

2019	Nomination for the CNRS Bronze medal
2016	Best paper award at ACM Genetic and Evolutionary Computation Conference (GECCO), leading conference in Evolutionary Computation, with J. Lengler (≈6.5% of accepted and ≈2.5% of submitted papers receive an award)
2014	Offer for an Independent Minerva Research Group Leadership Position (5 years, W2 equivalent) within the Max Planck Society (declined)
2013	Best paper award at GECCO, with B. Doerr and F. Ebel
2013	Otto Hahn Medal of the Max Planck Society (honoring ≈30 researchers across all scientific disciplines per year)
2012–13	Feodor Lynen PostDoc fellowship of the Alexander von Humboldt foundation
2012–13	Offers for highly selective PostDoc fellowships by the German Academic Exchange Service (DAAD) and École Polytechnique (both declined)
2012–14	Selected participant in the Fast Track Program of the Robert Bosch Foundation as only Computer Scientist among the 20 awardees
2012	Best paper award at GECCO, with B. Doerr
2010–12	Google Europe PhD Fellowship covering salary, travel costs, and other expenses
2010	Best paper award at GECCO, with B. Doerr and D. Johannsen
2004–07	Fellow of the Foundation of German Business (SDW), undergraduate stipend

Recent Project Leadership Positions

2016-20	Vice-chair of COST Action CA15140: Improving Applicability of Nature-Inspired Optimisation by Joining Theory and Practice (ImAppNIO), 31 participating countries, yearly budget ≈100k-150k€
2020-22	Optimization Meets Systems Biology (Opt4SysBio), Project funded by the Paris Ile-de-France Region, together with Benno Schwikowski from Institute Pasteur, 84k€
2019-21	Automated Algorithm Selection for Discrete Black-Box Optimization (AlgoSelect), Project funded by the Paris Ile-de-France Region, 55k€
2020-22	Theoretical Foundations of Dynamic Parameter Selection for Randomized Optimization Heuristics, joint CNRS - RFBR project, with Benjamin Doerr, Arina Buzdalova, and Maxim Buzdalov, 4k€ p.a.
2018-20	Online Configuration of Heuristic Optimization Algorithms, Project funded by the Paris Ile-de-France Region, 12k€
2019	Interactive Multi-Objective Optimization, project funded by the LIP6 laboratory, together with Thibaut Lust, 4k€
2014-21	Various PGMO/Labex Mathématique Hadamard projects, budget of 7k-15k€ p.a., complete list available on my homepage

Recent Grant Reviewing Services

2019-20	Marie Sklodowska-Curie Actions Innovative Training Networks (H2020-MSCA-ITN)
2020	Dutch Research Council, NWO, Veni grant scheme
2018	Austrian Science Fund (FWF), Erwin Schroedinger fellowship
2018	ALECS, a Marie Sklodowska-Curie COFUND fellowship program at Lero, the Irish Software Research Centre

Recent Hiring Committee Membership

2019	Member of the hiring committee for the position *Maître de conférences no. 25MCF1683 (0110) Profil: Probabilités et Applications* at the Mathematics department of Sorbonne University
2018	Member of the hiring committee for the position *Maître de conférences no. 25MCF2733 (0015) - Profil: Mathématiques fondamentales et interactions* at the Mathematics department of Sorbonne University

Editorial Responsibilities and Program Committees

2020	PC chair of PPSN 2020, with M. Emmerich and H. Trautmann (268 submissions)
since 2019	Associate Editor, ACM Transactions on Evolutionary Learning and Optimization
since 2019	Editorial Board member, Evolutionary Computation Journal (MIT press)
2019	PC chair of 15th ACM/SIGEVO Workshop on Foundations of Genetic Algorithms, with D. Arnold (31 submissions)
since 2018	Advisory Board Member of Springer's Natural Computing Series
2017	PC chair of the theory track at GECCO, with D. Sudholt (19 submissions)
2017–18	Guest editor of a special issue in Algorithmica, with D. Sudholt
2015	PC chair of the theory track at GECCO, with F. Chicano (19 submissions)
2015-17	Guest editor of a special issue in Algorithmica, with F. Chicano
since 2011	PC member in conferences on **evolutionary computation** [e.g., GECCO'11–'20, WCCI/CEC'13–'20, PPSN'12–'20, EvoCop'17–'21, EA'17–'19, SSCI'15–'20, FOGA'15–'21], **artificial intelligence** [AAAI'20, IJCAI'20-'21, ECAI'20, ECML-PKDD'20, LOD'20], **distributed computing** [IPDPS'17, ICDCD'15, FOMC'13], **computability** [[Computability in Europe 2021], and **general algorithm theory** [ESA'17 (track B)],

Recent Organization of Events and Mentoring

2021	Hot-off-the-Press chair at GECCO 2021
2020	Co-organization of the Benchmarked: Optimization Meets Machine Learning workshop at the Lorentz Center
2020	Co-organization of the Open Optimization Competition, with Facebook
2020	Co-organization of workshops on benchmarking at PPSN and at GECCO
since 2020	Member of the EC Technical Committee of the IEEE Computational Intelligence Society
since 2019	Founding member of the Benchmarking Network
2019	Co-organization of Dagstuhl seminar 19431 on Theory of Optimization Heuristics
2019	Late-Breaking Abstracts chair for GECCO and co-organizer of a GECCO workshop on Discrete Black-Box Optimization Benchmarking
2017	Local organization of COST training school (1 week, ≈40 participants, Paris)
2017	Co-organization of Dagstuhl seminar 17191 on Theory of Optimization Heuristics
2013–18	Mentor within the Minerva Program of the Max Planck Society
2016	Tutorial chair for PPSN, with N. Bredeche
2014–16	Co-organizer of the women@GECCO workshops

PhD Jury Membership

2020	Brieuc Guinard, Université de Paris, France
2018	Hao Wang, Leiden University, The Netherlands

Teaching Activities in Paris

Responsible for the course 2.24.2 on *Solving optimization problems with search heuristics* as part of the Parisian Master of Research in Computer Science (MPRI), 3 ECTS, 24 hours. I am teaching this course together with C. Dürr. The course has been held every year since 2015, but is pausing for one year in 2020/21.

Supervision of PostDoctoral Researchers

- Martin Krejca (01/21-09/22)
- Hao Wang (01/20 - 08/20), now Assistant Professor at Leiden University

Supervision of PhD Students

- Diederick Vermetten (since 11/19)
Integrated and Dynamic Algorithm Selection and Configuration
Leiden University, supervising together with Thomas Bäck

- Quentin Renau (since 02/19)
Landscape-Aware Search Heuristic for the Configuration of Radar Networks
École Polytechnique, supervising together with Benjamin Doerr and Johann Dreo
CIFRE thesis, with Thales Research & Technology

- Anja Jankovic (since 10/18)
Benefits of Adaptive Parameter Choices in Discrete Black-Box Optimization
Sorbonne Université, EDITE scholarship, main supervisor

- Furong Ye (since 10/17)
Benchmarking Discrete Optimization Heuristics
Leiden University, fellow of the Chinese scholarship council (CSC No. 201706310143), supervising together with Thomas Bäck

- Jing Yang (10/15–09/18)
Designing Superior Evolutionary Algorithms via Insights From Black-Box Complexity Theory
École Polytechnique, supervised together with Benjamin Doerr

Supervision of Undergraduate Students and PhD Interns

- Dominik Schröder (Leiden University, 07/2020-04/2021)
Title of the project: Dynamic algorithm selection for continuous black-box optimization
supervising together with Thomas Bäck and Hao Wang

- Amine Aziz-Alaoui (ISAE-SUPAERO, summer 2020)
Title of the project: Automated Algorithm Design using Exploratory Landscape Analysis
supervised together with Benjamin Doerr and Johann Dreo

- Andy Rabetafika (ISAE-SUPAERO, summer 2020)
Title of the project: Machine learning and discrepancy theory
supervised together with Benjamin Doerr and Johann Dreo

• Raphaël Cosson (MPRI, Paris Diderot Univ., summer 2019)
Title of the project: Online Configuration of Heuristic Optimization Algorithms
supervised together with Thomas Weise

• Vincent Aubry (MPRI, ENS, summer 2019)
Title of the project: Query Complexity of Mastermind
supervised together with Benjamin Doerr

• Anissa Kheireddine (Sorbonne University, summer 2019)
Title of the project: Dynamic Algorithm Configuration for Interactive Learning
supervised together with Thibaut Lust

• Diederick Vermetten (Leiden University, summer 2019)
Title of the project: Online Selection of CMA-ES Variants
supervised together with Thomas Bäck

• Nathan Buskulic (Sorbonne University, summer 2018)
Title of the project: Optimal Evolutionary Algorithms with Dynamic Parameters
Main supervisor

• Anja Jankovic (Sorbonne University, summer 2018)
Title of the project: Randomness in Scheduling
supervised together with Fanny Pascual and Nguyen Kim Thang

• Eduardo Carvalho Pinto (MPRI, Ecole Polytechnique, summer 2017)
Title of the project: Self-Adjusting Parameter Choices for Discrete Black-Box Optimization
Main supervisor

• Jing Yang (École Polytechnique, summer 2015)
Title of the project: Tight Bounds for the Unbiased Black-Box Complexity of OneMax
supervised together with Benjamin Doerr

• Axel de Perthuis de Laillevault (École Polytechnique, summer 2014)
Title of the project: Evolutionary Algorithms with Iterated Initial Sampling
supervised together with Benjamin Doerr

• Franziska Ebel (MPI, defended spring 2013)
Title of the thesis: Lessons from the Black-Box: Fast Crossover-Based Genetic Algorithms
supervised together with Benjamin Doerr

• G. Ramakrishna (PhD student intern at MPI, summer 2012)
Title of the project: Computing Minimum Cycle Bases in Graphs of Bounded Treewidth
supervised together with Jens M. Schmid

• Vijay Ingalalli (MPI, defended autumn 2011)
Title of the thesis: Evolutionary Algorithms to Compute Lower Bounds for the Star Discrepancy
supervised together with Benjamin Doerr

• Jong-Hyun Lee (MPI, winter 2011/12)
Title of the project: Playing Mastermind with Constant Size Memory
supervised together with Benjamin Doerr and Reto Spöhel

Bibliography

[AAD+19] Peyman Afshani, Manindra Agrawal, Benjamin Doerr, Carola Doerr, Kasper Green Larsen, and Kurt Mehlhorn. The query complexity of a permutation-based variant of Mastermind. *Discrete Applied Mathematics*, 260:28–50, 2019.

[AAG18] Youhei Akimoto, Anne Auger, and Tobias Glasmachers. Drift theory in continuous search spaces: expected hitting time of the $(1 + 1)$-ES with $1/5$ success rule. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'18)*, pages 801–808. ACM, 2018.

[ABD20] Denis Antipov, Maxim Buzdalov, and Benjamin Doerr. Fast mutation in crossover-based algorithms. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'20)*, pages 1268–1276. ACM, 2020.

[AD11] Anne Auger and Benjamin Doerr. *Theory of Randomized Search Heuristics*. World Scientific, 2011.

[AD20] Denis Antipov and Benjamin Doerr. Runtime analysis of a heavy-tailed $(1 + (\lambda, \lambda))$ genetic algorithm on jump functions. In *Proc. of Parallel Problem Solving from Nature (PPSN'20)*, volume 12270 of *LNCS*, pages 545–559. Springer, 2020.

[ADK19] Denis Antipov, Benjamin Doerr, and Vitalii Karavaev. A tight runtime analysis for the $(1 + (\lambda, \lambda))$ GA on LeadingOnes. In *Proc. of Foundations of Genetic Algorithms (FOGA'19)*, pages 169–182. ACM, 2019.

[ADK20] Denis Antipov, Benjamin Doerr, and Vitalii Karavaev. The $(1+(\lambda, \lambda))$ GA is even faster on multimodal problems. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'20)*, pages 1259–1267. ACM, 2020.

[AM16] Aldeida Aleti and Irene Moser. A systematic literature review of adaptive parameter control methods for evolutionary algorithms. *ACM Computing Surveys*, 49:56:1–56:35, 2016.

[Bäc92] Thomas Bäck. The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. In *Proc. of Parallel Problem Solving from Nature (PPSN'92)*, pages 87–96. Elsevier, 1992.

[Bäc96] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, 1996.

[Bäc98] Thomas Bäck. An overview of parameter control methods by self-adaption in evolutionary algorithms. *Fundam. Informaticae*, 35(1-4):51–66, 1998.

[BB19a] Anton Bassin and Maxim Buzdalov. The 1/5-th rule with rollbacks: on self-adjustment of the population size in the $(1+(\lambda, \lambda))$ GA. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'20, Companion Material)*, pages 277–278. ACM, 2019.

[BB19b] Nina Bulanova and Maxim Buzdalov. Limited memory, limited arity unbiased black-box complexity: first insights. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'19, Companion Material)*, pages 2020–2023. ACM, 2019.

[BB20] Anton Bassin and Maxim Buzdalov. The $(1+(\lambda,\lambda))$ genetic algorithm for permutations. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'20, Companion Material)*, pages 1669–1677. ACM, 2020.

[BD17] Maxim Buzdalov and Benjamin Doerr. Runtime analysis of the $(1 + (\lambda, \lambda))$ Genetic Algorithm on random satisfiable 3-CNF formulas. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'17)*, pages 1343–1350. ACM, 2017.

[BD19] Nathan Buskulic and Carola Doerr. Maximizing drift is not optimal for solving OneMax. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'19, Companion Material)*, pages 425–426. ACM, 2019. An extension of this work is to appear in the Evolutionary Computation journal.

[BD20] Maxim Buzdalov and Carola Doerr. Optimal mutation rates for the $(1 + \lambda)$ EA on OneMax. In *Proc. of Parallel Problem Solving from Nature (PPSN'20)*, volume 12270 of *LNCS*, pages 574–587. Springer, 2020.

[BDB+20] Thomas Bartz-Beielstein, Carola Doerr, Jakob Bossek, Sowmya Chandrasekaran, Tome Eftimov, Andreas Fischbach, Pascal Kerschke, Manuel López-Ibáñez, Katherine M. Malan, Jason H. Moore, Boris Naujoks, Patryk Orzechowski, Vanessa Volz, Markus Wagner, and Thomas Weise. Benchmarking in optimization: Best practice and open issues. *CoRR*, abs/2007.03488, 2020.

[BDDV20] Maxim Buzdalov, Benjamin Doerr, Carola Doerr, and Dmitry Vinokurov. Fixed-target runtime analysis. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'20)*, pages 1295–1303. ACM, 2020.

[BDK16] Maxim Buzdalov, Benjamin Doerr, and Mikhail Kever. The unrestricted black-box complexity of jump functions. *Evolutionary Computation*, 24(4):719–744, 2016.

[BDK20a] Jakob Bossek, Carola Doerr, and Pascal Kerschke. Initial design strategies and their effects on sequential model-based optimization: an exploratory case study based on BBOB. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'20)*, pages 778–786. ACM, 2020.

[BDK+20b] Jakob Bossek, Carola Doerr, Pascal Kerschke, Aneta Neumann, and Frank Neumann. Evolving sampling strategies for one-shot optimization tasks. In *Proc. of Parallel Problem Solving from Nature (PPSN'20)*, volume 12269 of *LNCS*, pages 111–124. Springer, 2020. Extended version available at https://arxiv.org/abs/1912.08956.

[BDN10] Süntje Böttcher, Benjamin Doerr, and Frank Neumann. Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In *Proc. of Parallel Problem Solving from Nature (PPSN'10)*, volume 6238 of *LNCS*, pages 1–10. Springer, 2010.

[BDR20] Arina Buzdalova, Carola Doerr, and Anna Rodionova. Hybridizing the 1/5-th success rule with q-learning for controlling the mutation rate of an evolutionary algorithm. In *Proc. of Parallel Problem Solving from Nature (PPSN'20)*, volume 12270 of *LNCS*, pages 485–499. Springer, 2020.

[BG02] Brian Beachkofski and Ramana Grandhi. Improved Distributed Hypercube Sampling. In *43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*. American Institute of Aeronautics and Astronautics, 2002.

[BLS13] Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, 2013.

[BLS14] Golnaz Badkobeh, Per Kristian Lehre, and Dirk Sudholt. Unbiased black-box complexity of parallel search. In *Proc. of Parallel Problem Solving from Nature (PPSN'14)*, volume 8672 of *LNCS*, pages 892–901. Springer, 2014.

[BMAB20] Anton Bouter, Stefanus C. Maree, Tanja Alderliesten, and Peter A. N. Bosman. Leveraging conditional linkage models in gray-box optimization with the real-valued gene-pool optimal mixing evolutionary algorithm. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'20)*, pages 603–611. ACM, 2020.

[BPS03] Mauro Birattari, Luis Paquete, and Thomas Stützle. Classification of metaheuristics and design of experiments for the analysis of components. https://www.researchgate.net/publication/2557723_Classification_of_Metaheuristics_and_Design_of_Experiments_for_the_Analysis_of_Components, 2003. Technical report.

[Bsh09] Nader H. Bshouty. Optimal algorithms for the coin weighing problem with a spring scale. In *Proc. of Conference on Learning Theory (COLT'09)*. Omnipress, 2009.

[BWB20] Rick Boks, Hao Wang, and Thomas Bäck. A modular hybridization of particle swarm optimization and differential evolution. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'20, Companion Material)*, pages 1418–1425. ACM, 2020.

[CCD+20] Marie-Liesse Cauwet, Camille Couprie, Julien Dehos, Pauline Luc, Jérémy Rapin, Morgane Riviere, Fabien Teytaud, Olivier Teytaud, and Nicolas Usunier. Fully parallel hyperparameter search: Reshaped space-filling. In *Proc. of International Conference on Machine Learning (ICML'20)*, pages 2749–2759, 2020.

[CCH96] Zhixiang Chen, Carlos Cunha, and Steven Homer. Finding a hidden code by asking questions. In *Proc. of Conference on Computing and Combinatorics (COCOON'96)*, pages 50–55. Springer, 1996.

[CD18] Eduardo Carvalho Pinto and Carola Doerr. A simple proof for the usefulness of crossover in black-box optimization. In *Proc. of Parallel Problem Solving from Nature (PPSN'18)*, volume 11102 of *LNCS*, pages 29–41. Springer, 2018.

[CFSS08] Luís Da Costa, Álvaro Fialho, Marc Schoenauer, and Michèle Sebag. Adaptive operator selection with dynamic multi-armed bandits. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'08)*, pages 913–920. ACM, 2008.

[Chv83] Vasek Chvátal. Mastermind. *Combinatorica*, 3:325–329, 1983.

[CL20] Brendan Case and Per Kristian Lehre. Self-adaptation in nonelitist evolutionary algorithms on discrete problems with unknown structure. *IEEE Transactions on Evolutionary Computation*, 24(4):650–663, 2020.

[CM66] David G. Cantor and William H. Mills. Determination of a subset from certain combinatorial properties. *Canadian Journal of Mathematics*, 18:42–48, 1966.

[CMT04] Sébastien Cahon, Nordine Melab, and El-Ghazali Talbi. Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics. *J. Heuristics*, 10(3):357–380, 2004. Latest release available on https://nojhan.github.io/paradiseo/.

[CSC+19] Borja Calvo, Ofer M. Shir, Josu Ceberio, Carola Doerr, Hao Wang, Thomas Bäck, and Jose A. Lozano. Bayesian performance analysis for black-box optimization benchmarking. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'19, Companion Material)*, pages 1789–1797. ACM, 2019.

[CSD20] Christian Leonardo Camacho-Villalón, Thomas Stützle, and Marco Dorigo. Grey wolf, firefly and bat algorithms: Three widespread algorithms that do not contain any novelty. In *Proc. of Swarm Intelligence (ANTS'20)*, volume 12421 of *LNCS*, pages 121–133. Springer, 2020.

[CSWA15] Francisco Chicano, Andrew M. Sutton, L. Darrell Whitley, and Enrique Alba. Fitness probability distribution of bit-flip mutation. *Evolutionary Computation*, 23(2):217–248, 2015.

[DD15] Benjamin Doerr and Carola Doerr. Optimal parameter choices through self-adjustment: Applying the 1/5-th rule in discrete settings. In *Proc. of Genetic and Evolutionary Computation Conference (GECO'15)*, pages 1335–1342. ACM, 2015.

[DD16a] Benjamin Doerr and Carola Doerr. The impact of random initialization on the runtime of randomized search heuristics. *Algorithmica*, 75(3):529–553, 2016.

[DD16b] Benjamin Doerr and Carola Doerr. Theory for non-theoreticians. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'15, Companion Material)*, pages 463–482. ACM, 2016.

[DD18] Benjamin Doerr and Carola Doerr. Optimal static and self-adjusting parameter choices for the (1+(λ,λ)) genetic algorithm. *Algorithmica*, 80:1658–1709, 2018.

[DD19] Nguyen Dang and Carola Doerr. Hyper-parameter tuning for the $(1 + (\lambda, \lambda))$ GA. In *Proc. of Genetic and Evolutionary Computation Conference (GECO'19)*, pages 889–897. ACM, 2019.

[DD20] Benjamin Doerr and Carola Doerr. Theory of parameter control mechanisms for discrete black-box optimization: Provable performance gains through dynamic parameter choices. In *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 271–321. Springer, 2020.

[DDE13] Benjamin Doerr, Carola Doerr, and Franziska Ebel. Lessons from the black-box: Fast crossover-based genetic algorithms. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'13)*, pages 781–788. ACM, 2013.

[DDE15] Benjamin Doerr, Carola Doerr, and Franziska Ebel. From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science*, 567:87 – 104, 2015.

[DDG18] Benjamin Doerr, Carola Doerr, and Michael Gnewuch. Probabilistic lower bounds for the discrepancy of latin hypercube samples. In *Contemporary Computational Mathematics - A Celebration of the 80th Birthday of Ian Sloan*, pages 339–350. Springer, 2018.

[DDK14] Benjamin Doerr, Carola Doerr, and Timo Kötzing. The unbiased black-box complexity of partition is polynomial. *Artificial Intelligence*, 216:275–286, 2014.

[DDK15] Benjamin Doerr, Carola Doerr, and Timo Kötzing. Unbiased black-box complexities of jump functions. *Evolutionary Computation*, 23(4):641–670, 2015.

[DDK18] Benjamin Doerr, Carola Doerr, and Timo Kötzing. Static and self-adjusting mutation strengths for multi-valued decision variables. *Algorithmica*, 80(5):1732–1768, 2018.

[DDL19] Benjamin Doerr, Carola Doerr, and Johannes Lengler. Self-adjusting mutation rates with provably optimal success rules. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'19)*, pages 1479–1487. ACM, 2019. Full version available at https://arxiv.org/abs/1902.02588.

[DDN19] Benjamin Doerr, Carola Doerr, and Frank Neumann. Fast re-optimization via structural diversity. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'19)*, pages 233–241. ACM, 2019.

[DDST16] Benjamin Doerr, Carola Doerr, Reto Spöhel, and Henning Thomas. Playing Mastermind with many colors. *J. ACM*, 63:42:1–42:23, 2016.

[DDY16] Benjamin Doerr, Carola Doerr, and Jing Yang. k-bit mutation with self-adjusting k outperforms standard bit mutation. In *Proc. of Parallel Problem Solving from Nature (PPSN'16)*, volume 9921 of *LNCS*, pages 824–834. Springer, 2016.

[DDY20] Benjamin Doerr, Carola Doerr, and Jing Yang. Optimal parameter choices via precise black-box analysis. *Theoretical Computer Science*, 801:1–34, 2020.

[Dev72] Luc Devroye. *The compound random search*. Ph.D. dissertation, Purdue Univ., West Lafayette, IN, 1972.

[DFK+16] Duc-Cuong Dang, Tobias Friedrich, Timo Kötzing, Martin S. Krejca, Per Kristian Lehre, Pietro S. Oliveto, Dirk Sudholt, and Andrew M. Sutton. Escaping local optima with diversity mechanisms and crossover. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'16)*, pages 645–652. ACM, 2016.

[DFK+18] Duc-Cuong Dang, Tobias Friedrich, Timo Kötzing, Martin S. Krejca, Per Kristian Lehre, Pietro S. Oliveto, Dirk Sudholt, and Andrew M. Sutton. Escaping local optima using crossover with emergent diversity. *IEEE Transactions on Evolutionary Computation*, 22(3):484–497, 2018.

[DGW14] Carola Doerr, Michael Gnewuch, and Magnus Wahlström. Calculation of discrepancy measures and applications. In William Chen, Anand Srivastav, and Giancarlo Travaglini, editors, *A Panorama of Discrepancy Theory*, pages 621–678. Springer, 2014.

[DGWY19] Benjamin Doerr, Christian Gießen, Carsten Witt, and Jing Yang. The $(1 + \lambda)$ evolutionary algorithm with self-adjusting mutation rate. *Algorithmica*, 81(2):593–631, 2019.

[DHK12] Benjamin Doerr, Edda Happ, and Christian Klein. Crossover can provably be useful in evolutionary computation. *Theoretical Computer Science*, 425:17–33, 2012.

[DJK+11] Benjamin Doerr, Daniel Johannsen, Timo Kötzing, Per Kristian Lehre, Markus Wagner, and Carola Winzen. Faster black-box algorithms through higher arity operators. In *Proc. of Foundations of Genetic Algorithms (FOGA'11)*, pages 163–172. ACM, 2011.

[DJK+13] Benjamin Doerr, Daniel Johannsen, Timo Kötzing, Frank Neumann, and Madeleine Theile. More effective crossover operators for the all-pairs shortest path problem. *Theoretical Computer Science*, 471:12–26, 2013.

[DJS+13] Benjamin Doerr, Thomas Jansen, Dirk Sudholt, Carola Winzen, and Christine Zarges. Mutation rate matters even when optimizing monotonic functions. *Evolutionary Computation*, 21(1):1–27, 2013.

[DJW02] Stefan Droste, Thomas Jansen, and Ingo Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276:51–81, 2002.

[DJW06] Stefan Droste, Thomas Jansen, and Ingo Wegener. Upper and lower bounds for randomized search heuristics in black-box optimization. *Theory of Computing Systems*, 39:525–544, 2006.

[DJW12] Benjamin Doerr, Daniel Johannsen, and Carola Winzen. Multiplicative drift analysis. *Algorithmica*, 64:673–697, 2012.

[DKLW13] Benjamin Doerr, Timo Kötzing, Johannes Lengler, and Carola Winzen. Black-box complexities of combinatorial problems. *Theoretical Computer Science*, 471:84–106, 2013.

[DL16a] Duc-Cuong Dang and Per Kristian Lehre. Self-adaptation of mutation rates in non-elitist populations. In *Proc. of Parallel Problem Solving from Nature (PPSN'16)*, volume 9921 of *Lecture Notes in Computer Science*, pages 803–813. Springer, 2016.

[DL16b] Carola Doerr and Johannes Lengler. The (1+1) elitist black-box complexity of LeadingOnes. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'16)*, pages 1131–1138. ACM, 2016.

[DL17a] Carola Doerr and Johannes Lengler. Introducing elitist black-box models: When does elitist behavior weaken the performance of evolutionary algorithms? *Evolutionary Computation*, 25, 2017.

[DL17b] Carola Doerr and Johannes Lengler. OneMax in black-box models with several restrictions. *Algorithmica*, 78:610–640, 2017.

[DL18] Carola Doerr and Johannes Lengler. The (1+1) elitist black-box complexity of LeadingOnes. *Algorithmica*, 80:1579–1603, 2018.

[DLMN17] Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen. Fast genetic algorithms. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'17)*, pages 777–784. ACM, 2017.

[DN20] Benjamin Doerr and Frank Neumann. *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*. Springer, 2020.

[Doe16] Benjamin Doerr. Optimal parameter settings for the $(1 + (\lambda, \lambda))$ genetic algorithm. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'16)*, pages 1107–1114. ACM, 2016.

[Doe19] Benjamin Doerr. Analyzing randomized search heuristics via stochastic domination. *Theor. Comput. Sci.*, 773:115–137, 2019.

[Doe20a] Benjamin Doerr. Probabilistic tools for the analysis of randomized optimization heuristics. In *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 1–87. Springer, 2020.

[Doe20b] Carola Doerr. Complexity theory for black-box optimization heuristics. In *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 133–212. Springer, 2020.

[Dor92] Marco Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, 1992.

[DP09] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomised Algorithms*. Cambridge University Press, 2009.

[DP10] J. Dick and F. Pillichshammer. *Digital Nets and Sequences*. Cambridge University Press, 2010.

[dPdLDD15] Axel de Perthuis de Laillevault, Benjamin Doerr, and Carola Doerr. Money for nothing: Speeding up evolutionary algorithms through better initialization. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'15)*, pages 815–822. ACM, 2015.

[DR13] Carola Doerr and François-Michel De Rainville. Constructing low star discrepancy point sets with genetic algorithms. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'13)*, pages 789–796. ACM, 2013.

[DS90] G. Dueck and T. Scheuer. Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. *J. Comput. Phys.*, 90:161–175, 1990.

[DT09] Benjamin Doerr and Madeleine Theile. Improved analysis methods for crossover-based algorithms. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'09)*, pages 247–254. ACM, 2009.

[DW12] Benjamin Doerr and Carola Winzen. Black-box complexity: Breaking the $O(n \log n)$ barrier of LeadingOnes. In *Artificial Evolution (EA'11), Revised Selected Papers*, volume 7401 of *LNCS*, pages 205–216. Springer, 2012.

[DW14a] Benjamin Doerr and Carola Winzen. Playing Mastermind with constant-size memory. *Theory of Computing Systems*, 55:658–684, 2014.

[DW14b] Benjamin Doerr and Carola Winzen. Ranking-based black-box complexity. *Algorithmica*, 68:571–609, 2014.

[DW14c] Benjamin Doerr and Carola Winzen. Reducing the arity in unbiased black-box complexity. *Theoretical Computer Science*, 545:108–121, 2014.

[DW18a] Carola Doerr and Markus Wagner. Sensitivity of parameter control mechanisms with respect to their initialization. In *Proc. of Parallel Problem Solving from Nature (PPSN'18)*, volume 11102 of *LNCS*, pages 360–372. Springer, 2018.

[DW18b] Carola Doerr and Markus Wagner. Simple on-the-fly parameter selection mechanisms for two classical discrete black-box optimization benchmark problems. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'18)*, pages 943–950. ACM, 2018.

[DWY18a] Benjamin Doerr, Carsten Witt, and Jing Yang. Runtime analysis for self-adaptive mutation rates. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'18)*, pages 1475–1482. ACM, 2018.

[DWY+18b] Carola Doerr, Hao Wang, Furong Ye, Sander van Rijn, and Thomas Bäck. IOHprofiler: A Benchmarking and Profiling Tool for Iterative Optimization Heuristics. *CoRR*, abs/1810.05281, 2018. Available at http://arxiv.org/abs/1810.05281.

[DWY+18c] Carola Doerr, Hao Wang, Furong Ye, Diederick Vermetten, and Thomas Bäck. IOHprofiler github repository. https://github.com/IOHprofiler, 2018.

[DYH+20] Carola Doerr, Furong Ye, Naama Horesh, Hao Wang, Ofer M. Shir, and Thomas Bäck. Benchmarking discrete optimization heuristics with iohprofiler. *Applied Soft Computing*, 88:106027, 2020.

[DYvR+18] Carola Doerr, Furong Ye, Sander van Rijn, Hao Wang, and Thomas Bäck. Towards a theory-guided benchmarking suite for discrete black-box optimization heuristics: profiling $(1 + \lambda)$ EA variants on OneMax and LeadingOnes. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'18)*, pages 951–958. ACM, 2018.

[EHM99] Agoston Endre Eiben, Robert Hinterding, and Zbigniew Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3:124–141, 1999.

[ER63] Paul Erdős and Alfréd Rényi. On two problems of information theory. *Magyar Tudományos Akadémia Matematikai Kutató Intézet Közleményei*, 8:229–243, 1963.

[ES03] A. E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, 2003.

[FCSS08] Álvaro Fialho, Luís Da Costa, Marc Schoenauer, and Michèle Sebag. Extreme value based adaptive operator selection. In *Proc. of Parallel Problem Solving from Nature (PPSN'08)*, volume 5199 of *LNCS*, pages 175–184. Springer, 2008.

[FCSS10] Álvaro Fialho, Luís Da Costa, Marc Schoenauer, and Michèle Sebag. Analyzing bandit-based adaptive operator selection mechanisms. *Annals of Mathematics and Artificial Intelligence*, 60:25–64, 2010.

[FT11] Hervé Fournier and Olivier Teytaud. Lower bounds for comparison based evolution strategies using vc-dimension and sign patterns. *Algorithmica*, 59:387–408, 2011.

[FW04] Simon Fischer and Ingo Wegener. The Ising model on the ring: Mutation versus recombination. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'04)*, volume 3102 of *LNCS*, pages 1113–1124. Springer, 2004.

[GK00] Vladimir Grebinski and Gregory Kucherov. Optimal reconstruction of graphs under the additive model. *Algorithmica*, 28:104–124, 2000.

[Glo86] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533 – 549, 1986. Applications of Integer Programming.

[Gol72] M. Golay. A class of finite binary sequences with alternate auto-correlation values equal to zero (corresp.). *IEEE Transactions on Information Theory*, 18(3):449–450, 1972.

[Goo09] Michael T. Goodrich. On the algorithmic complexity of the Mastermind game with black-peg results. *Information Processing Letters*, 109:675–678, 2009.

[GP14] Brian W. Goldman and William F. Punch. Parameter-less population pyramid. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'14)*, pages 785–792. ACM, 2014.

[GSW09] Michael Gnewuch, Anand Srivastav, and Carola Winzen. Finding optimal volume subintervals with k points and calculating the star discrepancy are NP-hard problems. *Journal of Complexity*, 25:115–127, 2009.

[GW17] Christian Gießen and Carsten Witt. The interplay of population size and mutation probability in the $(1 + \lambda)$ EA on OneMax. *Algorithmica*, 78(2):587–609, 2017.

[GWW12] Michael Gnewuch, Magnus Wahlström, and Carola Winzen. A new randomized algorithm to approximate the star discrepancy based on threshold accepting. *SIAM J. Numerical Analysis*, 50:781–807, 2012.

[HAB+16] Nikolaus Hansen, Anne Auger, Dimo Brockhoff, Dejan Tusar, and Tea Tusar. COCO: performance assessment. *CoRR*, abs/1605.03560, 2016.

[HAB20] Nikolaus Hansen, Anne Auger, and Dimo Brockhoff. Data repository of the COCO benchmark environment [HAR+20]. https://coco.gforge.inria.fr/data-archive/, 2010-2020.

[Hal60] John H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2:84–90, 1960.

[HAR+20] Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. COCO: a platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, pages 1–31, 2020.

[HBS19] Naama Horesh, Thomas Bäck, and Ofer M. Shir. Predict or screen your expensive assay: Doe vs. surrogates in experimental combinatorial optimization. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'19)*, pages 274–284. ACM, 2019.

[HFRA09] Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions. Technical Report RR-6829, INRIA, 2009.

[HJJ03] Darrall Henderson, Sheldon H. Jacobson, and Alan W. Johnson. The theory and practice of simulated annealing. In Fred Glover and Gary A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 287–319. Springer, 2003.

[HO01] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[HOS20] George T. Hall, Pietro S. Oliveto, and Dirk Sudholt. Analysis of the performance of algorithm configurators for search heuristics with global mutation operators. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'20)*, pages 823–831. ACM, 2020.

[HPR+18] Hsien-Kuei Hwang, Alois Panholzer, Nicolas Rolin, Tsung-Hsi Tsai, and Wei-Mei Chen. Probabilistic analysis of the (1+1)-evolutionary algorithm. *Evolutionary Computation*, 26, 2018.

[HW19] Hsien-Kuei Hwang and Carsten Witt. Sharp bounds on the runtime of the (1+1) EA via drift analysis and analytic combinatorial tools. In *Proc. of Conference on Foundations of Genetic Algorithms (FOGA'19)*, pages 1–12. ACM, 2019.

[HWHC13] Doug Hains, L. Darrell Whitley, Adele E. Howe, and Wenxiang Chen. Hyperplane Initialized Local Search for MAXSAT. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'13)*, pages 805 – 812. ACM, 2013.

[Jan13] Thomas Jansen. *Analyzing Evolutionary Algorithms—The Computer Science Perspective*. Springer, 2013.

[JD20] Anja Jankovic and Carola Doerr. Landscape-aware fixed-budget performance regression and algorithm selection for modular CMA-ES variants. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'20)*, pages 841–849. ACM, 2020.

[JM02] David Stifler Johnson and Lyle A. McGeoch. Experimental Analysis of Heuristics for the STSP. In *The Traveling Salesman Problem and its Variations*, volume 12 of *Combinatorial Optimization*, chapter 9, pages 369 – 443. Kluwer Academic Publishers, 2002.

[JP11] Gerold Jäger and Marcin Peczarski. The number of pessimistic guesses in generalized black-peg Mastermind. *Information Processing Letters*, 111:933–940, 2011.

[JSW98] Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

[JW02] Thomas Jansen and Ingo Wegener. The analysis of evolutionary algorithms - a proof that crossover really can help. *Algorithmica*, 34:47–66, 2002.

[JZ11] Thomas Jansen and Christine Zarges. Analysis of evolutionary algorithms: from computational complexity analysis to algorithm engineering. In *Proc. of Foundations of Genetic Algorithms (FOGA'11)*, pages 1–14. ACM, 2011.

[JZ14] Thomas Jansen and Christine Zarges. Performance analysis of randomised search heuristics operating with a fixed budget. *Theoretical Computer Science*, 545:39–58, 2014.

[KE95] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proc. of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[KGV83] Scott Kirkpatrick, C. D. Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[KHE15] Giorgos Karafotias, Mark Hoogendoorn, and A.E. Eiben. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, 19:167–187, 2015.

[KHNT19] Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1):3–45, 2019.

[KLLG19] Robert Kleinberg, Kevin Leyton-Brown, Brendan Lucier, and Devon R. Graham. Procrastinating with confidence: Near-optimal, anytime, adaptive algorithm configuration. In *Proc. of Advances in Neural Information Processing Systems (NeurIPS'19)*, pages 8881–8891, 2019.

[KLR+11] Johannes W. Kruisselbrink, Rui Li, Edgar Reehuis, Jeroen Eggermont, and Thomas Bäck. On the log-normal self-adaptation of the mutation rate in binary search spaces. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'11)*, pages 893–900. ACM, 2011.

[KMH+04] Stefan Kern, Sibylle D. Müller, Nikolaus Hansen, Dirk Büche, Jiri Ocenasek, and Petros Koumoutsakos. Learning probability distributions in continuous evolutionary algorithms - a comparative review. *Natural Computing*, 3:77–112, 2004.

[KT19] P. Kerschke and H. Trautmann. Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning. *Evolutionary Computation*, 27(1):99–127, 2019.

[Lad05] Véronique Ladret. Asymptotic hitting time for a simple evolutionary model of protein folding. *Journal of Applied Probability*, 42:39–51, 2005.

[LDC+16] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[Len18] Johannes Lengler. A general dichotomy of evolutionary algorithms on monotone functions. In *Proc. of Parallel Problem Solving from Nature (PPSN'18)*, volume 11102 of *LNCS*, pages 3–15. Springer, 2018.

[Len20] Johannes Lengler. Drift analysis. In Benjamin Doerr and Frank Neumann, editors, *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 89–131. Springer, Cham, 2020.

[LFKZ14] Ke Li, Álvaro Fialho, Sam Kwong, and Qingfu Zhang. Adaptive operator selection with bandits for a multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 18:114–130, 2014.

[LGW+20] Q. Liu, W. V. Gehrlein, L. Wang, Y. Yan, Y. Cao, W. Chen, and Y. Li. Paradoxes in numerical comparison of optimization algorithms. *IEEE Transactions on Evolutionary Computation*, 24(4):777–791, 2020.

[Lin64] Bernt Lindström. On a combinatory detection problem i. *Mathematical Institute of the Hungarian Academy of Science*, 9:195–207, 1964.

[Lin65] Bernt Lindström. On a combinatorial problem in number theory. *Canadian Mathematical Bulletin*, 8:477–490, 1965.

[LL02] Pedro Larrañaga and José Antonio Lozano, editors. *Estimation of Distribution Algorithms*. Genetic Algorithms and Evolutionary Computation. Springer, 2002.

[LLM07] Fernando G. Lobo, Cláudio F. Lima, and Zbigniew Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*. Springer, 2007.

[LM20] Johannes Lengler and Jonas Meier. Large population sizes and crossover help in dynamic environments. In *Proc. of Parallel Problem Solving from Nature (PPSN'20)*, volume 12269 of *LNCS*, pages 610–622. Springer, 2020.

[LS18] Johannes Lengler and Angelika Steger. Drift analysis and evolutionary algorithms revisited. *Combinatorics, Probability & Computing*, 27(4):643–666, 2018.

[LS19] Per Kristian Lehre and Dirk Sudholt. Parallel black-box complexity with tail bounds. *CoRR*, abs/1902.00107, 2019.

[LW10] Per Kristian Lehre and Carsten Witt. Black-box search by unbiased variation. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'10)*, pages 1441–1448. ACM, 2010.

[LW12] Per Kristian Lehre and Carsten Witt. Black-box search by unbiased variation. *Algorithmica*, 64:623–642, 2012.

[MA19] Daiki Morinaga and Youhei Akimoto. Generalized drift analysis in continuous domain: linear convergence of (1 + 1)-ES on strongly convex functions with lipschitz continuous gradients. In *Proc. of Foundations of Genetic Algorithms (FOGA'19)*, pages 13–24. ACM, 2019.

[Mat09] Jiří Matoušek. *Geometric Discrepancy*. Springer, Berlin, 2nd edition, 2009.

[MBC79] Michael D. McKay, Richard J. Beckman, and William J. Conover. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics*, 21:239–245, 1979.

[MDRT20] Laurent Meunier, Carola Doerr, Jérémy Rapin, and Olivier Teytaud. Variance reduction for better sampling in continuous domains. In *Proc. of Parallel Problem Solving from Nature (PPSN'20)*, volume 12269 of *LNCS*, pages 154–168. Springer, 2020.

[MH97] N. Mladenović and P. Hansen. Variable neighborhood search. *Comput. Oper. Res.*, 24(11):10971100, 1997.

[MP96] H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions i. binary parameters. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Proc. of Parallel Problem Solving from Nature (PPSN'96)*, pages 178–187. Springer, 1996.

[MSKH15] Mario A. Muñoz, Yuan Sun, Michael Kirley, and Saman K. Halgamuge. Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges. *Inf. Sci.*, 317:224–245, 2015.

[MT10] Robin A. Moser and Gábor Tardos. A constructive proof of the general lovász local lemma. *Journal of the ACM*, 57(2):11:1–11:15, 2010.

[Müh92] Heinz Mühlenbein. How genetic algorithms really work: Mutation and hillclimbing. In *Proc. of Parallel Problem Solving from Nature (PPSN'92)*, pages 15–26. Elsevier, 1992.

[Nie92] Harald Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*. Society for Industrial and Applied Mathematics, 1992.

[NW10] Frank Neumann and Carsten Witt. *Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity*. Springer, 2010.

[PM16] Tom Packebusch and Stephan Mertens. Low autocorrelation binary sequences. *Journal of Physics A: Mathematical and Theoretical*, 49(16):165001, 2016.

[RABD19] Anna Rodionova, Kirill Antonov, Arina Buzdalova, and Carola Doerr. Offspring population size matters when comparing evolutionary algorithms with self-adjusting mutation rates. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'19)*, pages 855–863. ACM, 2019.

[Rec73] Ingo Rechenberg. *Evolutionsstrategie*. Friedrich Fromman Verlag (Günther Holzboog KG), Stuttgart, 1973.

[RT18] Jérémy Rapin and Olivier Teytaud. Nevergrad - A gradient-free optimization platform. https://GitHub.com/FacebookResearch/Nevergrad, 2018.

[Rud97] Günter Rudolph. *Convergence Properties of Evolutionary Algorithms*. Kovac, 1997.

[RV11] Jonathan Rowe and Michael Vose. Unbiased black box search algorithms. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'11)*, pages 2035–2042. ACM, 2011.

[SEBB18] Jörg Stork, Ágoston Endren Eiben, and Thomas Bartz-Beielstein. A new taxonomy of continuous global optimization algorithms, 2018. arXiv e-prints:1808.08818.

[SF96] Jim Smith and Terence C. Fogarty. Self adaptation of mutation rates in a steady state genetic algorithm. In *Proc. of IEEE International Conference on Evolutionary Computation*, pages 318–323. IEEE, 1996.

[Sob67] Ilya Meyerovich Sobol. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86–112, January 1967.

[Sör15] Kenneth Sörensen. Metaheuristics - the metaphor exposed. *International Transactions in Operational Research (ITOR)*, 22:3–18, 2015.

[SP97] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[SS68] Michael A. Schumer and Kenneth Steiglitz. Adaptive step size random search. *IEEE Transactions on Automatic Control*, 13:270–276, 1968.

[Sud05] Dirk Sudholt. Crossover is provably essential for the Ising model on trees. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'05)*, pages 1161–1167. ACM, 2005.

[Sud12] Dirk Sudholt. Crossover speeds up building-block assembly. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'12)*, pages 689–702. ACM, 2012.

[Sud13] Dirk Sudholt. A new method for lower bounds on the running time of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 17:418–435, 2013.

[Sud17] Dirk Sudholt. How crossover speeds up building block assembly in genetic algorithms. *Evolutionary Computation*, 25(2):237–274, 2017.

[TB20] Dirk Thierens and Peter A. N. Bosman. Model-based evolutionary algorithms: GECCO 2020 tutorial. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'20, Companion Material)*, pages 590–619. ACM, 2020.

[TG06] Olivier Teytaud and Sylvain Gelly. General lower bounds for evolutionary algorithms. In *Proc. of Parallel Problem Solving from Nature (PPSN'06)*, volume 4193 of *LNCS*, pages 21–31. Springer, 2006.

[Thi05] Dirk Thierens. An adaptive pursuit strategy for allocating operator probabilities. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'05)*, pages 1539–1546. ACM, 2005.

[vLA87] Peter J.M. van Laarhoven and Emilie H. L. Aarts. *Simulated Annealing: Theory and Applications*. Springer, 1987.

[vRWvLB16] Sander van Rijn, Hao Wang, Matthijs van Leeuwen, and Thomas Bäck. Evolving the structure of evolution strategies. In *Proc. of IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2016.

[VWBD20] Diederick Vermetten, Hao Wang, Thomas Bäck, and Carola Doerr. Towards dynamic algorithm selection for numerical black-box optimization: investigating BBOB as a use case. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'20)*, pages 654–662. ACM, 2020.

[VWDB20] Diederick Vermetten, Hao Wang, Carola Doerr, and Thomas Bäck. Integrated vs. sequential approaches for selecting and tuning CMA-ES variants. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'20)*, pages 903–912. ACM, 2020.

[WCL+14] Thomas Weise, Raymond Chiong, Jörg Lässig, Ke Tang, Shigeyoshi Tsutsui, Wenxiang Chen, Zbigniew Michalewicz, and Xin Yao. Benchmarking optimization algorithms: An open source framework for the traveling salesman problem. *IEEE Computational Intelligence Magazine*, 9(3):40–52, 2014.

[Win11] Carola Winzen. *Toward a complexity theory for randomized search heuristics*. PhD thesis, Universität Saarbrücken, 2011.

[Wit13] Carsten Witt. Tight bounds on the optimization time of a randomized search heuristic on linear functions. *Combinatorics, Probability & Computing*, 22:294–318, 2013.

[WW18] Thomas Weise and Zijun Wu. Difficult features of combinatorial optimization problems and the tunable w-model benchmark problem for simulating them. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'18, Companion Material)*, pages 1769–1776. ACM, 2018.

[Yao77] Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proc. of Foundations of Computer Science (FOCS'77)*, pages 222–227. IEEE, 1977.

[YDB19] Furong Ye, Carola Doerr, and Thomas Bäck. Interpolating local and global search by controlling the variance of standard bit mutation. In *Proc. of IEEE Congress on Evolutionary Computation (CEC'19)*, pages 2292–2299. IEEE, 2019.