Theoretical Foundations of Artificial Immune Systems

Dissertation

zur Erlangung des Grades eines

D o k t o r s d e r N a t u r w i s s e n s c h a f t e n

der Technischen Universität Dortmund
an der Fakultät für Informatik
von

Christine Zarges

Dortmund
2011

Tag der mündlichen Prüfung:	05.07.2011
Dekanin:	Prof. Dr. Gabriele Kern-Isberner
Gutachter:	Dr. Thomas Jansen und Prof. Dr. Günter Rudolph

Acknowledgements

First of all I would like to express my gratitude to my advisor Thomas Jansen, who has always been a great support. In particular, I am very grateful that he continued my supervision after moving to Cork and gave me the opportunity to closely work together with him during several research stays.

I thank my co-authors Benjamin Doerr, Thomas Jansen, Dirk Sudholt, Christian Thyssen, and Carola Winzen. Moreover, I thank Jon Timmis and Giuseppe Nicosia for discussing promising research questions that are of interest for the artificial immune systems community.

I gratefully acknowledge the financial support from several institutions: the Federal Office for Information Security (BSI) within the project 'Evolutionary algorithms for autonomous systems', Google by the 'Google Anita Borg Memorial Scholarship', and last but not least the MPI Saarbrücken.

Finally I thank my parents for their steady support.

Contents

Part I.

Prologue

1. Foreword

Artificial immune systems[1] (AIS) are a special class of biologically inspired algorithms, which are based on the immune system of vertebrates (Dasgupta 1998; Dasgupta and Niño 2008; de Castro and Timmis 2002a). The field constitutes a relatively new and emerging area of research in Computational Intelligence that has achieved various promising results in different areas of application (Hart and Timmis 2008), e. g., learning, classification, anomaly detection, and (function) optimization. In this thesis, we concentrate on optimization applications of AIS. An increasing and often stated problem of the field is the lack of a theoretical basis for AIS (Timmis 2006, 2007; Timmis et al. 2008c) as most work so far only concentrated on the direct application of immune principles. Timmis et al. (2008c) give an overview on theoretical work existing prior to this thesis. It can easily be recognized that with respect to optimization, the work done so far mainly covers convergence analysis. Hence, the following citation provides the main motivation for the work done within this thesis.

> *Thus a major theoretical challenge for the future is to give sharper bounds for the performance of an IA (or of other clonal selection mechanisms) when applied to some specific functions, and to see how this depends on the form of the mutation operator and other features of the algorithm. Ideally, one would like to have an idea of which type of IA would be most effective for a particular class of problems.* (Timmis et al. 2008c)

To the best of our knowledge this thesis constitutes the first rigorous run time analyses of immune-inspired operators and thus adds substantially to the demanded theoretical foundation of AIS. We consider two very common aspects of AIS. On one hand, we provide a theoretical analysis for different hypermutation operators frequently employed in AIS. On the other hand, we examine a popular diversity mechanism named aging. We compare our findings with corresponding results from the analysis of other nature-inspired randomized search heuristics, in particular evolutionary algorithms. Moreover, we focus on the practical implications of our theoretical results in order to bridge the gap between theory and practice. Therefore, we derive guidelines for parameter settings and point out typical situations where certain concepts seem promising. These analyses contribute to the understanding of how AIS actually work and in which applications they excel other randomized search heuristics.

[1]also known as immunological computation (IC), immunocomputing, computational immunology, immune algorithms (IA), or immune-based systems

1.1. Overview

The structure of this thesis is as follows. We start with a short introduction to AIS in Chapter 2. We describe common immune principles that are used as inspiration for AIS, embed AIS into the broader context of general randomized search heuristics and establish the notation for the rest of the thesis. Afterwards, the two main analytical parts of the thesis follow, dealing with mutation (Part II) and aging (Part III) in AIS, respectively. Both parts contain an introductory chapter presenting related work, the contribution of this thesis as well as the analytical framework used in the subsequent chapters. The theoretical findings are accompanied by empirical studies. We describe the contents of these remaining chapters in more detail.

In Part II we first consider concrete mutation operators used in existing AIS, namely inversely fitness-proportional mutation (Chapter 4) and contiguous somatic hypermutations (Chapter 5). Since in contrast to, e. g., evolutionary algorithms, in AIS typically high mutation probabilities are used, we take a broader view in the last chapter of this part (Chapter 6) and analyze the general effects larger mutation probabilities can have.

In Part III we first examine the most important parameter of aging, namely the maximal lifespan (Chapter 8), and derive guidelines for setting this parameter. Using these results, we compare a commonly used aging operator from AIS with a similar operator known from evolutionary algorithms (Chapter 9) by pointing out strength and weaknesses of both approaches. Moreover, a third aging operator is introduced that provably shares the advantages of these two aging mechanisms. In the last chapter of this part (Chapter 10) we establish a more structured view on aging as used in AIS by considering different aging and replacement strategies and the role of age diversity for effective aging operators.

Part IV deals with the relevance of our theoretical findings for practical applications. We point out the importance of experiments for bridging the gap between theory and practice, present related work and review relevant findings from the preceding chapters of the thesis (Chapter 11). Afterwards, we discuss limitations of the common approach used to determine the optimization time of a randomized search heuristics by considering analyses from this thesis as well as from other publications (Chapter 12).

Finally, we summarize the findings presented in this thesis and discuss directions for future work in Chapter 13. Moreover, an overview of notations and the mathematical tools used can be found in the appendix.

1.2. Underlying Publications

This thesis is based on the following publications, listed in the order in which they appear here. For all joint papers with k authors, this author's contribution can be quantified as at least $1/k$.

1. Christine Zarges (2008): Rigorous runtime analysis of inversely fitness proportional mutation rates. In Günter Rudolph, Thomas Jansen, Simon Lucas, Carlo Poloni,

and Nicola Beume, editors, *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature (PPSN X)*, volume 5199 of *Lecture Notes in Computer Science*, pages 112–122. Springer. *Best Student Paper Award.*

The results in Chapter 4.2, Chapter 4.3.1 and in parts Chapter 4.5 are based on this publication.

2. Christine Zarges (2009): On the utility of the population size for inversely fitness proportional mutation rates. In Ivan Garibay, Thomas Jansen, R. Paul Wiegand, and Annie S. Wu, editors, *Proceedings of the 10th ACM SIGEVO Conference on Foundations of Genetic Algorithms (FOGA 2009)*, pages 39–46. ACM Press.

   Chapter 4.3.2 is based on this publication.

3. Thomas Jansen and Christine Zarges (2011a): Analyzing different variants of immune inspired somatic contiguous hypermutations. *Theoretical Computer Science*, 412(6):517–533.

   Chapter 5 is based on this article. Parts of this publication are also published in the following conference article.

   - Thomas Jansen and Christine Zarges (2009a): A theoretical analysis of immune inspired somatic contiguous hypermutations for function optimization. In Paul S. Andrews, Jon Timmis, Nick D.L. Owens, Uwe Aickelin, Emma Hart, Andrew Hone, and Andy M. Tyrrell, editors, *Proceedings of the 8th International Conference on Artificial Immune Systems (ICARIS 2009)*, volume 5666 of *Lecture Notes in Computer Science*, pages 80–94. Springer.

4. Benjamin Doerr, Thomas Jansen, Dirk Sudholt, Carola Winzen, and Christine Zarges (2011): Mutation rate matters even when optimizing monotone functions. Submitted.

   Chapter 6 is based on this publication. Parts of this publication are also published in the following conference article.

   - Benjamin Doerr, Thomas Jansen, Dirk Sudholt, Carola Winzen, and Christine Zarges (2010a): Optimizing monotone functions can be difficult. In Robert Schaefer, Carlos Cotta, Joanna Kołodziej, and Günter Rudolph, editors, *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature (PPSN XI)*, volume 6238 of *Lecture Notes in Computer Science*, pages 42–51. Springer.

5. Christian Horoba, Thomas Jansen, and Christine Zarges (2009): Maximal age in randomized search heuristics with aging. In Günther R. Raidl, editor, *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO 2009)*, pages 803–810. ACM Press. *Nominated for a Best Paper Award in the track "Genetic Algorithms".*

   Chapter 8 is based on this publication.

6. Thomas Jansen and Christine Zarges (2011b): On benefits and drawbacks of aging strategies for randomized search heuristics. *Theoretical Computer Science*, 412(6): 543–559.

   Chapter 9 is based on this article. Parts of this publication are also published in the following conference article.

   - Thomas Jansen and Christine Zarges (2009b): Comparing different aging operators. In Paul S. Andrews, Jon Timmis, Nick D.L. Owens, Uwe Aickelin, Emma Hart, Andrew Hone, and Andy M. Tyrrell, editors, *Proceedings of the 8th International Conference on Artificial Immune Systems (ICARIS 2009)*, volume 5666 of *Lecture Notes in Computer Science*, pages 95–108. Springer.

7. Thomas Jansen and Christine Zarges (2011c): On the role of age diversity for effective aging operators. *Evolutionary Intelligence*, 4(2):99–125.

   Chapter 10 is based on this article. Parts of this publication are also published in the following two conference articles.

   - Thomas Jansen and Christine Zarges (2010a): Aging beyond restarts. In Juergen Branke, editor, *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO 2010)*, pages 705–712. ACM Press.
   - Thomas Jansen and Christine Zarges (2010b): On the benefits of aging and the importance of details. In Emma Hart, Chris McEwan, Jon Timmis, and Andy Hone, editors, *Proceedings of the 9th International Conference on Artificial Immune Systems (ICARIS 2010)*, volume 6209 of *Lecture Notes in Computer Science*, pages 61–74. Springer.

8. Thomas Jansen and Christine Zarges (2011d): Analysis of evolutionary algorithms: From computational complexity analysis to algorithm engineering. In Hans-Georg Beyer and William B. Langdon, editors, *Proceedings of the 11th ACM SIGEVO Conference on Foundations of Genetic Algorithms (FOGA 2011)*, pages 1–14. ACM Press.

   Chapter 12 is based on this publication.

2. Preliminaries

The field of AIS comprises two main branches: on one hand immune modeling or mathematical theoretical immunology, which is closely related to immunology and aims at understanding the natural immune system by means of mathematics and computer science, in particular by supporting experimental analyses of the immune system (Timmis et al. 2008a,b); on the other hand engineering and optimization, which is concerned with problem solving by immune-inspired methods. In the latter aspect of AIS researchers aim at capturing certain properties of the natural immune system such as self-organization, learning, classification and adaptation capabilities, diversity, robustness and scalability in their problem solutions (de Castro and Timmis 2002a).

In this thesis, we concentrate on the computer science perspective and additionally restrict ourselves to optimization applications of AIS. In this context, a common definition of AIS can be stated as follows.

> *Artificial immune systems (AIS) are adaptive systems, inspired by theoretical immunology and observed immune functions, principles and models, which are applied to problem solving.* (de Castro and Timmis 2002a)

From this definition, we can deduce simple minimal requirements for an AIS. Clearly, just using immunology terminology does not qualify a system to be classified as an AIS. First, an AIS has to incorporate a model of a basic immune component, e. g. an immune cell. Second, it has to be modeled after immune principles derived from theoretical or experimental immunology. Last, it has to be designed for the purpose of problem solving. However, most AIS only use a few and often rather abstract immune-inspired concepts.

Even though anomaly detection, classification and learning are among the most natural applications for AIS they are also often applied to the problem of (function) optimization. According to de Castro and Timmis (2002a), the field of AIS was initiated by works of Farmer et al. (1986) and Hoffmann (1986). Farmer et al. (1986) suggested that computational intelligence methods could benefit from incorporating immune-inspired principles whereas Hoffmann (1986) executed a comparative study of the nervous and the immune system which inspired other researchers to investigate novel network models. With respect to problem solving, the first known applications were published in the early 1990s. However, this work was still well rooted within the immunology community and was in particular concerned with, at that time, recently introduced immune network models describing the maintenance of immune memory (Bersini 1992; Bersini and Varela 1991a,b) and learning abilities of the immune system (Cooke and Hunt 1995) as well as applications of immune principles in computer security (Forrest and Perelson 1991; Forrest et al. 1994, 1997; Kephart 1994). Later, algorithms for function optimization (de Castro and Von Zuben 2002b) and data clustering were developed (de Castro and Von Zuben 2000, 2002a). Concurrently, in 2002 the International Conference on Artificial Immune Systems[1] (ICARIS) was established. Since the field of AIS has become rather large, an exhaustive survey of applications and developments is clearly out of the scope of this thesis. However, there are several good textbook and review papers available (Dasgupta 1998; Dasgupta and Niño 2008; de Castro and Timmis 2002a; Flower and Timmis 2007; Hart and Timmis 2008). Moreover, a regularly updated bibliography can be found online (Dasgupta 2007). An overview on different AIS approaches for optimization was presented by Bernardino and Barbosa (2009).

As already mentioned in Chapter 1 there is a clear lack of theoretical work for AIS since much work so far concentrated on the direct application of certain immune principles (Timmis et al. 2008c). With respect to (function) optimization the work done prior to this thesis mainly covers convergence analysis (Clark et al. 2005; Clark 2008; Cutello et al. 2007c; Villalobos-Arias et al. 2004). We discuss some of these previous results when introducing the corresponding algorithms.

In the rest of this chapter, we aim at giving a broad overview of the field of AIS, its applications and some basic algorithms. In particular, we point out the different sources of inspiration from the natural immune system. We see that in contrast to other nature-inspired search heuristics, AIS derive from various immunological theories, for example the clonal selection principle, negative selection, immune networks or the danger theory. Since we concentrate on the analysis of AIS from a computational point of view the biological background used as a motivation is not too important to us. Therefore, only a rather abstract and focused introduction to the natural immune system is given. The interested reader is referred to standard textbooks on immunology for a more in-depth view on the biology side of the field (Delves et al. 2006; Flower and Timmis 2007; Janeway et al. 2005).

We concentrate on the aspect of applying AIS to optimization problems. In this context, current state-of-the-art AIS are in some sense very similar to other nature-inspired search heuristics, like e. g., evolutionary algorithms, ant colony optimization, and simulated annealing. We point out similarities and differences of these approaches and establish a common notation that is used throughout the rest of this thesis. Note, that parts of the following explanation is based on the corresponding chapters in de Castro and Timmis (2002a), Dasgupta and Niño (2008), and Flower and Timmis (2007).

2.1. Immunity-Based Computational Models and Algorithms

The main task of the immune system is to recognize and remove pathogens invading the organism. In this context, the immune system needs to be capable to distinguish between so-called *self*, i. e., the body's own cells, and *nonself* cells. This is somehow equivalent to solving a two-class classification problem. Note, that in immunology this is also known as the *self/nonself discrimination problem* or the *self/nonself theory*.

[1]`http://www.artificial-immune-systems.org`

The immune system is organized in a multilevel fashion. First, there are physical and chemical barriers like skin or saliva. Second, an immune response is separated into *innate* and *adaptive* mechanisms. Hereby, the innate immune system (Janeway and Medzhitov 2002) is non-specific, i. e., it reacts to general pathogens by recognizing molecular patterns that are found in microorganisms. It is responsible for the initiation and regulation of immune responses. In contrast to that, the adaptive (acquired) immune system (Janeway et al. 2005) is directed against specific pathogens. It enables the immune system to learn to recognize new pathogens, i. e., via a first infection or vaccination, and to memorize infections in order to speed up future immune reactions to similar pathogens (*secondary immune response*). Clearly, the immune system has to ensure that, by mistake, no self cells are classified as nonself, in order to prevent autoimmunity.

It is important that the immune system maintains a certain degree of diversity within its immune cells so that it is able to react to a large number of different pathogens. To this end, a large number of immune cells circulates through the organism. There are many different types of immune cells where *leukocytes* (white blood cells) are the most important ones. The class of leukocytes can be further divided into *lymphocytes*, which mediate the adaptive immune response, and *granulocytes* as well as *macrophages* as part of the innate immune system. From the computer science perspective, the adaptive immune system has caught most interest though recently reasearchers are also trying to develop systems based on the innate immune system (see Section 2.1.5). However, within this thesis we stick to algorithms inspired by mechanisms of the adaptive immune system (see Section 2.1.3).

There are essentially two types of lymphocytes: *T cells* and *B cells*. Basically, these two types of lymphocytes play different roles in the immune system though they may interact with each other. The primary lymphoid organs, namely the *bone marrow* and the *thymus gland*, are responsible for the permanent production and maturation of new immune cells. It is agreed that T cells are produced within the bone marrow and mature in the thymus gland whereas B cells develop and mature within the bone marrow. The secondary lymphoid organs, i. e., lymph vessels, lymph nodes and lymphoid follicles in tonsils, spleen, skin, and others, provide the transportation mechanism for immune cells and the environment for the interaction between lymphocytes and the molecular patterns of invading pathogens (*antigens*). Here, adaptive immune responses and the associated production of *antibodies* are initiated.

There are several immunological theories that try to explain the above mentioned properties of the immune system. After introducing the basic immune recognition mechanism and the corresponding representation concept in AIS, we discuss some of those immune principles as well as their applications and several algorithmic approaches. These algorithms serve as building blocks for engineering AIS. Some of the presented processes are controversial from an immunological point of view. However, since we are mainly interested in the computational perspective this is not critical to us.

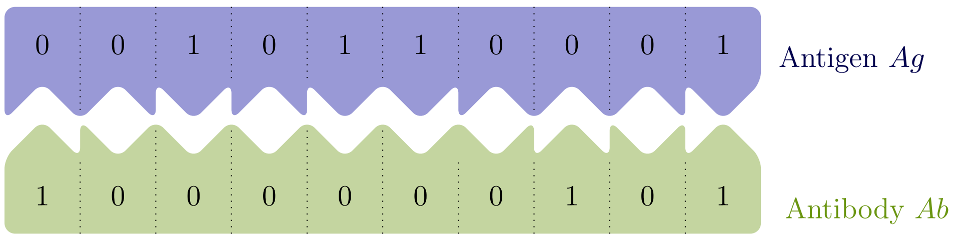


Figure 2.1.: Example of a binary Hamming shape space with $n = 10$ and Hamming distance $d(Ag, Ab) = 5$

2.1.1. Shape Spaces and Affinity Measures

Following de Castro and Timmis (2002a) we use a general terminology by referring to any type of cell receptor that is able to bind to a molecular pattern as *antibody*. Moreover, the molecular pattern itself is denoted as an *antigen*. The *shape space model* (Perelson and Oster 1979) was introduced to model and evaluate the interaction, i. e., binding, between antigens and antibodies. It is a well-known fact that an antigen is only recognized by an antibody if the two molecules bind complementary with each other over a considerable part of their surfaces.

Different features such as chemical properties and charge distributions influence the interaction between the molecules. These features are known as the *generalized shape* of a molecule. If the properties of a molecule can be described by a set of n parameters, the generalized shape space corresponds to a point $x = (x[0], \ldots, x[n-1])$ in an n-dimensional space S, called *shape space*. Moreover, the attributes can have different, problem-dependent types. According to these types, we distinguish between specific forms of shape spaces, e. g., real-valued ($x \in \mathbb{R}^n$), integer ($x \in \mathbb{Z}^n$), Hamming ($x \in \Sigma^n$ with $|\Sigma| = k$ for some constant k), or symbolic, where x is composed of different, possibly symbolic types. Figure 2.1 illustrates two generalized shapes over a binary Hamming shape space with $n = 10$, i. e., $S = \{0, 1\}^{10}$.

The strength of the binding between an antigen and an antibody is called *binding affinity* or simply *affinity*. In general it is evaluated via a distance measure $d : S \times S \to \mathbb{R}_0^+$ where the distance values should be proportional to the affinity. Clearly, this distance measure needs to be suitable for the chosen shape space. Typical choices are for example Euclidean or Manhattan distance for real-valued shape spaces or Hamming distance for Hamming shape spaces. A more detailed overview on different variants of the shape space model is given by Ji and Dasgupta (2007), Dasgupta and Niño (2008; Chapter 3) and de Castro and Timmis (2002a; pp. 62). The affinity for the example in Figure 2.1 is $d(Ag, Ab) = 5$ using the Hamming distance.

We still need to discuss the recognition of an antigen by an antibody in this model. For this purpose an *activation threshold* ε is defined and we say that the antibody Ab recognizes the antigen Ag if $d(Ag, Ab) \geq \varepsilon$. This corresponds to the observation that the binding of two molecules has to cover a considerable part of their surfaces. Moreover, we can control the number of antigens recognized by a single antibody by increasing or decreasing ε since this influences the size of the *recognition region* of an antibody within the considered shape space.

While the presented shape space model reflects the recognition of antigens and thus, is directly applicable for pattern recognition, classification and learning tasks, we need to reconsider our model when applying AIS to function optimization. In this context, antibodies often represent candidate solution of the function to be optimized and affinity corresponds to the objective function value for the solution. This can be seen as evaluating the affinity of an antibody within the external environment that is represented by the objective function rather than evaluating the interaction with other antibodies. Note, that this view is equivalent to the concept of *fitness* in evolutionary algorithms. We see that there exist AIS using both concepts of affinity when discussing immune networks in Section 2.1.4.

2.1.2. Negative Selection

Negative selection describes the maturation process of T cells in the thymus gland. As already mentioned immature T cells are produced within the bone marrow. Afterwards they are subject to a *censoring* process in the thymus where T cells reacting with self cells are removed in order to prevent autoimmunity. The remaining T cells leave the thymus and are added to the repertoire of immune cells in the immune system. We see that by means of negative selection recognition of patterns that are not part of a known set is performed. Note, that this is contrary to *positive selection* where the set of known pattern is recognized. Moreover, only one of the two classes of the classification problem needs to be known which is generally different for classical classification algorithms. This way, the immune system is enabled to react to so far unknown threats.

Forrest et al. (1994) used this theory as inspiration for a simple negative selection algorithm applied to the problem of change detection. In a first step, the set of elements to be monitored, i. e., the self set, is encoded using an appropriate shape space representation and affinity measure. Afterwards, a set of detectors is generated according to the negative selection process described above. Using this detector set, we can now monitor the considered system by constantly matching detectors against elements from the self set or in general from a set to be protected. If during this monitoring process a match occurs, a change in the system is detected. The algorithm by Forrest et al. (1994) is depicted in Figure 2.2.

There have been several further approaches for negative selection algorithms in the field of computer security (Ji and Dasgupta 2007). With respect to efficiency and accuracy of those algorithms the process of generating (a sufficient number of) detectors as well as an appropriate choice of the representation and affinity measure is crucial (Ji and Dasgupta 2007; Timmis et al. 2008c). Recently, new efficient approaches for the generation of detectors were presented (Elberfeld and Textor 2011; Liskiewicz and Textor 2010). As negative selection algorithms are mainly used for pattern recognition, classification and learning tasks, we do not further consider these algorithms within this thesis. However, since negative selection algorithms are among the first and most popular approaches in AIS, they constitute a major building block of the field.

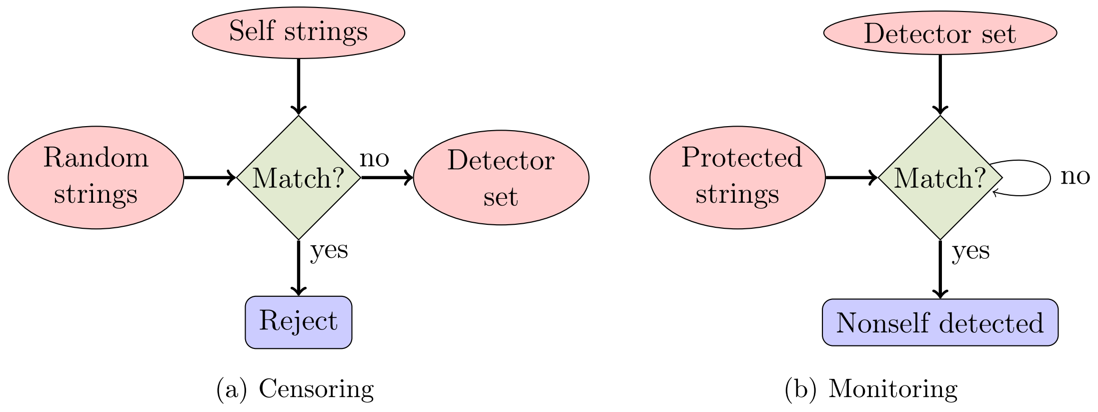


Figure 2.2.: Negative selection (adapted from Forrest et al. (1994))

2.1.3. Clonal Selection Principle

The *clonal selection principle* (Burnet 1959) describes the basic features of an adaptive immune response. Since the immune system aims at preserving a certain degree of diversity, only a small number of immune cells is able to recognize the antigen of an invading pathogen. Only immune cells, which recognize an antigen, proliferate (*clonal expansion*) and differentiate into clone- or antibody-producing cells. Some of those cells are retained in the immune memory to enable a fast secondary immune response. Clonal selection effects both, T cells and B cells but with the main difference that B cells are subject to mutation (*affinity maturation*) during reproduction whereas T cells are not. Due to these mutations at high rate (*hypermutation*) and a strong selective pressure, the affinity of the B cell clones with the antigen is increased. Due to its adaptive nature, usually clonal selection of B cells is used as inspiration for clonal selection algorithms. It is claimed that genetic algorithms without crossover are very similar to clonal selection algorithms (Forrest et al. 1993).

An important feature of clonal selection is the way cloning and mutation take place. There is strong evidence that the cloning and mutation probability of a B cell is related to its affinity with the antigen. In the community this is called fitness-proportional cloning rate and inversely fitness-proportional mutation probability, respectively, even though this is not true in a strict mathematical sense. It means that ‘the higher the affinity the higher the clone rate, and vice versa’. The same holds for inversely fitness-proportional mutation meaning ‘the higher the affinity the smaller the mutation probability, and vice versa’.

De Castro and Von Zuben (2002b) proposed the very popular clonal selection algorithm CLONALG following these basic ideas. CLONALG was initially applied to pattern recognition but was soon adapted to the problem of multi-modal function optimization. It is claimed that inversely fitness-proportional mutation is in particular suitable for multi-modal problems since each B cell is subject to an individual mutation probability, i. e., B cells with higher fitness, e. g., near peaks of the fitness landscape, are only mutated

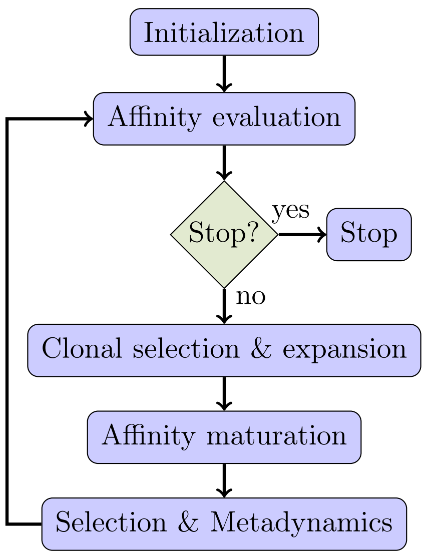


Figure 2.3.: Basic steps of CLONALG (de Castro and Von Zuben 2002b).

at low rate while B cells with small fitness are subject to high mutation probabilities. Note, that this is somehow similar to the concept of self-adaptation in evolutionary computation. To the best of our knowledge CLONALG was the first clonal selection algorithm for the purpose of function optimization.

The basic steps of CLONALG are depicted in Figure 2.3. After a random initialization, in each iteration some B cells are selected for cloning and mutation. Afterwards, the best B cells are selected for the next iteration. Moreover, the constant production of new random B cells within the bone marrow is simulated by replacing some B cells with low affinity by new random ones (*metadynamics*). Remember that in optimization applications of AIS the objective function or fitness value takes the role of affinity (compare Section 2.1.1). Thus, we speak of (inversely) fitness-proportional clone and mutation probabilities. However, for optimization it is suggested to select each B cell with equal probability for cloning instead of using a fitness-proportional cloning rate. We analyze the concrete mutation probability used in CLONALG within Chapter 4 of this thesis when discussing different variants of inversely fitness-proportional mutation probabilities.

At present, there is a variety of different AIS to tackle optimization problems that are built on the clonal selection principle. An overview of different approaches can be found in Brownlee (2007). Probably, the most established ones besides CLONALG are opt-IA (Cutello et al. 2004a), the B-Cell algorithm (Kelsey and Timmis 2003), and MISA (Coello Coello and Cortés 2005). These algorithms share a common approach in a broad sense but they differ in the concrete instantiation of the clonal selection principle.

The name opt-IA comprises several clonal selection algorithms following similar ideas. The approach was first presented by Cutello and Nicosia (2002a) under the name Immunological Algorithm. It is constantly further developed and applied to different optimization problems. It was renamed several times and is currently known as opt-IA. Other names are, e. g., Simple Immune Algorithm (SIA), Cloning Information Gain Aging (CLIGA), and Optimization Immune Algorithm (opt-IMMALG). For an extensive overview see Brownlee (2007).

The main idea of these algorithms is to calculate a *mutation potential M* that determines the maximal number of positions in the point representation to be mutated. Different types of such mutation potentials were developed, e. g., inversely fitness-proportional, fitness-proportional, or constant ones (Cutello et al. 2004a). Moreover, a *hypermacromutation* operator was proposed, where only a random contiguous region of the point is effected by this mutation operator. Another feature of the algorithm is the integration of an additional selection mechanism called *aging* that removes points, which exceed a pre-defined maximal lifespan, i. e., the maximal number of iterations a specific immune cell is allowed within the repertoire. Deterministic and stochastic variants of aging are known.

Cutello et al. (2005b) compare CLONALG and opt-IA empirically on different optimization problems. Cutello et al. (2007c) present a convergence analysis for a general immune algorithm that incorporates ideas of opt-IA. We analyze deterministic aging as used in opt-IA in Part III of this thesis.

The B-Cell algorithm (Kelsey and Timmis 2003) uses an alternative model called *somatic contiguous hypermutation* for the mutation probability in the affinity maturation process (Lamlum et al. 1999). Here, a contiguous region of the point representation is randomly selected and each position is mutated with a certain probability. This mutation probability is independent of the fitness. A Markov chain model and corresponding convergence results for the B-Cell algorithm are available (Clark et al. 2005; Clark 2008). We analyze somatic contiguous hypermutations in Chapter 5 of this thesis.

Finally, MISA (Coello Coello and Cortés 2005) constitutes an example for a multi-objective clonal selection algorithm. Since we only deal with single-objective optimization here, this approach is out of the scope of this thesis. A convergence analysis for MISA was presented by Villalobos-Arias et al. (2004).

2.1.4. Immune Network Theory

Due to the immune network model by Jerne (1974) the immune system can be described as a regulated network of cells, also called *idiotypic network*. The immune cells in the network interact by recognizing each other even if no antigen is present in their environment (known as *network stimulation* and *suppression*). Thus, immune cells are not isolated but rather communicate with each other. This is a fundamental difference to the clonal selection principle where it is assumed that the immune system is only stimulated in the presence of an antigen and at rest otherwise. It leads to the conclusion that self-tolerance, learning, and memory are rather global properties of the whole immune system than properties of single immune cells and that the recognition of an antigen is part of the dynamic behavior of the whole system. The dynamics of the system are disturbed in the presence of an antigen. Changes of the network due to this disturbance lead the network towards a mapping of the antigenic environment. However, among im-

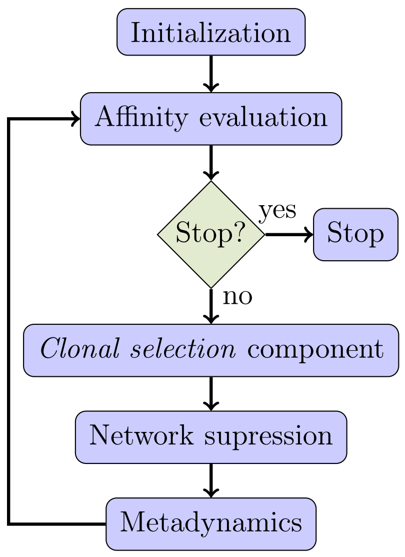


Figure 2.4.: Basic steps of opt-aiNet (de Castro and Timmis 2002b).

munologists, the immune network theory is discussed rather controversially (de Castro and Timmis 2002a; Langman and Cohn 1986; Stepney et al. 2005).

Two types of network models are distinguished: *continuous network models* based on differential equations and *discrete network models* based on difference equations and adaptation mechanisms. While the idea for the discrete model is derived from the continuous one, both models have been applied to learning and optimization, respectively. However, originally the continuous model was mainly used to simulate the immune system whereas discrete models were devoted to problem solving. Thus, we concentrate on discrete models here. A comparison of different immune network models was published by Galeano et al. (2005).

Probably, the most popular discrete immune network algorithm in the context of problem solving is aiNet (Artificial Immune NETwork) by de Castro and Von Zuben (2002a). Like CLONALG it was originally presented as a network learning algorithm, but there also exists an adaption for multi-modal function optimization, named opt-aiNet (de Castro and Timmis 2002b), which is in some sense quite similar to the above described clonal selection algorithms. It is also based on clonal selection ideas, but adds interaction mechanisms from the immune network theory.

The basic steps of opt-aiNet are depicted in Figure 2.4. After a random initialization, a kind of ‘clonal selection component’ is executed until the network has reached a stable state, which is determined via the average fitness of the cells in the network. During each iteration of the clonal selection step a constant number of clones is created for each cell and mutated inversely proportional to the fitness of this cell. The mutated clone with the highest fitness is introduced into the network.

Once the network has reached a stable state, *network suppression* is performed. Here, cells interact with each other by calculating their pairwise affinities. Remember, that affinity between cells corresponds to their distance. Thus, low affinity cells are removed from the network in order to preserve a certain level of diversity. Note, that this is done in an elitist way since in each affected pair of cells, the cell with the lower fitness is removed. Afterwards, new random cells are inserted into the network (*metadynamics*). This way, the number of B cells in the network is dynamically adapted. The concrete mutation probability in opt-aiNet is different from that in CLONALG. We also analyze this variant within this thesis (Chapter 4) when discussing inversely fitness-proportional mutation probabilities.

It is easy to see that opt-aiNet only incorporates network suppression. Another important feature of many immune network algorithm is the concept of stimulation. The *stimulation level* quantifies the interaction of the components in the immune network. Here, basically, two aspects are taken into account: on one hand, the number of antigens recognized by a given antibody or even the whole system, and on the other hand, the degree of interaction with the other antibodies in the network. It is used to determine the selection of cells for expansion and survival within the network. Moreover, it can (analogously to the affinity of a cell to an antigen in the clonal selection) influence the clone and mutation probability of a selected cell. Altogether, the structure of most network models can be described by the following equation due to Perelson (1989) where network stimulation and suppression describe the network dynamics while new elements and death of elements are part of the general immune metadynamics (de Castro and Timmis 2002a):

$$\begin{aligned}\text{Rate of population variation} = {} & \text{Network stimulation} - \text{Network suppression} \\ & + \text{Influx of new elements} - \text{Death of unstimulated elements}\end{aligned}$$

2.1.5. Other Approaches

The immune principles described in the foregoing sections are the most classical and popular ones for engineering AIS. In recent years, researchers have incorporated other mechanisms and theories from immunology into their algorithms and developed completly new approaches. Probably, the most important one among these is the *danger theory*[2] (Aickelin and Cayzer 2002; Matzinger 1994) that exhibits an alternative approach to the self/nonself discrimination of negative selection in the field of computer security. However, as the immune network theory, the danger theory is controversial (Aickelin et al. 2003; Langman and Cohn 2000; Vance 2000).

The danger theory claims that an immune response is triggered by the presence of *safe* and *danger signals* rather than the recognition of nonself cells since not all foreign antigens are considered dangerous. Examples for danger signals are signals from damaged cells such as tissue damage. In this process, *dendritic cells* play an important role. They are part of the innate immune system and act as a mediator between the innate and the adaptive immune system by detecting safe and danger signals. This way, they decide

[2] `http://www.dangertheory.com`

on whether the environment is to be considered safe or dangerous and thus, control the initiation of an immune response.

The *Dendritic Cell Algorithm* (DCA) is the largest research project within the danger theory branch of AIS. The original version of the DCA was proposed as part of the Danger Project (Aickelin et al. 2003). The first presentation of a prototype originates from Greensmith et al. (2005). Later, a real-time implementation was presented (Greensmith et al. 2006). The DCA is still further developed and analyzed.

Strongly related to the danger theory are ideas taken from the innate immune system. Twycross and Aickelin (2005) present a first concept for a framework using mechanisms known from innate immunity. The first real system named 'libtissue' build on these ideas originates from Twycross and Aickelin (2006). There are also approaches to combine ideas from both concepts, innate as well as adaptive immunity, to enhance the developed systems (Tedesco et al. 2006).

Finally, there is an increasing number of hybrid algorithms, combining ideas from immunology with other nature-inspired algorithms, e. g., neural networks or evolutionary algorithms. For an overview on these approaches and other modern immune models see de Castro and Timmis (2002a; Chapter 5–6) and Dasgupta and Niño (2008; Chapter 6).

2.2. Artificial Immune Systems as Randomized Search Heuristics

General randomized search heuristics (RSHs) are a class of algorithms used in practical settings where there is no time or expertise to develop problem-specific algorithms. They provide a powerful and flexible way of tackling different problems and are hoped to be efficient on a broad class of problems.

There are many different search heuristics like randomized local search (Michiels et al. 2007) or tabu search (Glover and Laguna 1997). Each heuristic implements some general idea of how search should be conducted. These ideas are often borrowed from other fields. They are for example inspired by

- natural processes, like simulated annealing (Aarts et al. 2005) that implements the process of annealing in metallurgy in an abstract way,
- biological processes, like evolutionary algorithms (EAs), mimicking the process of natural evolution (de Jong 2006), or
- biological systems, like artificial neural networks (ANN), inspired by natural neural networks such as the human nervous system (Rojas 1996), ant colony optimization (ACO), based on the behavior of foraging ants (Dorigo and Stützle 2004), and of course AIS, modeled after the immune systems of vertebrates (Dasgupta and Niño 2008; de Castro and Timmis 2002a).

Although these randomized search heuristics derive from quite different paradigms they can share some general ideas and exhibit certain similarities. However, despite these similarities they do differ in the concrete implementations of the concepts and thus, an implementation deriving from one specific paradigm may appear senseless in another one.

General randomized search heuristics are typically very easy to implement and apply. But it turns out to be extremely challenging to prove in a rigorous way results about their performance and limitations since they exhibit a complex random behavior (Rabinovich et al. 1992; Witt 2009). While the general idea is to apply a randomized search heuristic 'right out of the box' in practice it is almost always necessary to adjust the randomized search heuristic to the concrete problem at hand by adding more complex mechanisms and modifying the search strategies to achieve acceptable performance. Moreover, in practice, it makes sense to combine ideas from different randomized search heuristics in order to improve the performance of the algorithm. Clearly, this increased complexity complicates the task of a theoretical analysis even more.

Since it is highly desirable to obtain a clear understanding of the working principles and properties of the different operators employed it makes sense to study their effects in isolation. Such analyses serve as building blocks of a theory of randomized search heuristics. We follow this approach for the analyses executed in this thesis. Furthermore, we perform the analyses on instructive example functions in order to highlight specific properties of the operators under consideration. Such example functions allow for the comparison of different heuristics and serve as building blocks for a useful theory since they can be considered as stepping stones for further analyses. Moreover, they can be used as counter examples for disproving common assumptions. In our analyses we use on one hand well-known example functions for which a large number of existing results for comparison are available. On the other hand we use carefully constructed example functions that exhibit paradigmatic properties in order to make our points. This is a common approach in the theoretical analysis of randomized search heuristics.

The theoretical analysis of general randomized search heuristics is a modern and important area of research. While for other randomized search heuristics, in particular for evolutionary algorithms, there is a growing body of useful analytical tools and relevant results (Auger and Doerr 2011; Droste et al. 2006; Neumann and Witt 2010; Oliveto et al. 2007; Oliveto and Witt 2010; Wegener 2003), the analyses of AIS is, as mentioned above, still in its infancy. In this thesis, we provide rigorous analyses for different immune-inspired operators by adapting analytical tools from evolutionary algorithms. We compare our findings with results for other randomized search heuristics when available. Moreover, we derive guidelines for parameter settings and point out typical situations where certain concepts seem promising. These analyses contribute to the understanding of how AIS actually work and in which applications they excel over other randomized search heuristics.

2.3. A General Framework and Notations

While being applicable in very different situations one of the most important tasks of randomized search heuristics is (function) optimization. In this thesis, we restrict ourselves to this application. We consider the case where for the search space $\{0,1\}^n$ of bit strings of fixed length n an objective function $f\colon \{0,1\}^n \to \mathbb{R}$ is given and a globally optimal search point, i. e., an arbitrary point from

$$\mathrm{OPT} = \left\{x \in \{0,1\}^n \mid f(x) = \max\left\{f(x') \mid x' \in \{0,1\}^n\right\}\right\},$$

is sought. We adopt the notion of *fitness* for the value $f(x)$ from evolutionary algorithms. Since randomized search heuristics are most often implemented on standard binary computers the restriction to $\{0,1\}^n$ is not fundamental. Clearly, concentrating on maximization is no restriction since maximizing f is equivalent to minimizing $-f$. Note, however, that sometimes we additionally require f to be positive. We point out those further restrictions where necessary.

The algorithms under consideration follow the common scheme known from evolutionary algorithms as shown in Figure 2.5 and are in this respect very similar to these kind of randomized search heuristics. They use a collection of search points C of size μ and work in rounds where in each round λ new search point $y_1, \dots, y_\lambda \in \{0,1\}^n$ are generated as random variation of existing search points. Afterwards μ search points are selected for the next round. In this phase, additional mechanisms to preserve a certain degree of diversity within the collection of search points can be applied (see e. g. Friedrich et al. (2009b) for the analysis of common mechanisms from the field of evolutionary algorithms). Note, that within this thesis we stick to the case $\lambda = 1$. For the selection for reproduction we simply use uniform selection. The selection for replacement follows basically the idea of plus-selection where the search points for the next iteration are chosen from the set of the μ old and the λ new search points. We denote the elements of C by x_i, i. e., $C = \{x_1, x_2, \dots, x_\mu\}$, and the i-th bit of a search point x by $x[i]$, i. e., $x = (x[0], \dots, x[n-1])$. We abbreviate the number of 1-bits in a bit string x by $|x|_1$ and the number of 0-bits by $|x|_0$. We sometimes denote the result of a mutation of a search point x by $\mathrm{mut}(x)$ and the result from a variation and a subsequent selection by x^+, e. g., for $y = \mathrm{mut}(x)$ we have $x^+ = y$ if $f(y) \geq f(x)$ and $x^+ = x$ otherwise. The Hamming distance between two search points x and y is denoted by $H(x, y)$.

As usual in the analysis of general randomized search heuristics, there is no stopping criterion in the algorithm and we investigate the first point of time when a global optimum of f is reached. This equals the number of iterations until a global optimum is found and we denote this as the *optimization time* of the algorithm. Note, that this model does not count the cost of initialization. However, since in this thesis the optimization time is always larger than μ, this is not relevant with respect to asymptotic results. In some algorithms, the number of iterations also corresponds to the number of addtionally function evaluations, which is a common measure for the run time of randomized search heuristics since function evaluations are assumed to be the most costly operations of the randomized search heuristic while other operations tend to be algorithmically simple and can be carried out relatively quickly. It is important to notice that the algorithm itself does not know that it has found an optimum. We discuss limitations of this approach in Chapter 12 of this thesis.

We denote by $T_{A,f}$ the optimization time of Algorithm A on some function f. Moreover, let $T_{A,f,C}$ denote its optimization time when the algorithm is started with some

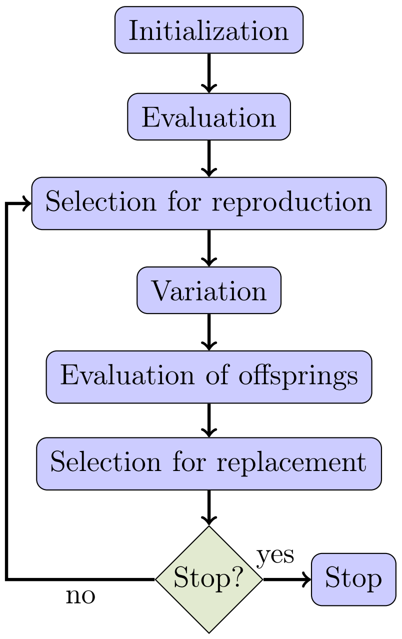


Figure 2.5.: General schema of evolutionary algorithms.

specific initial collection of search points $C = \{x_1, x_2, \ldots, x_\mu\}$ instead of using random initialization. Since the optimization time $T_{A,f}$ is a random variable we investigate its mean $E(T_{A,f})$ as well as sometimes information on its distribution like $\text{Prob}\,(T_{A,f} < t)$ or $\text{Prob}\,(T_{A,f} > t)$ for relevant points of time t. We say that an event A occurs *with high probability (w. h. p.)* if $\text{Prob}\,(A) = 1 - o(1)$ and *with overwhelming probability (w. o. p.)* if $\text{Prob}\,(A) = 1 - 2^{-\Omega(n)}$. For the optimization times, we make use of common asymptotic notation (Definition B.2) where all asymptotics are with respect to the bit string length n. We call a bound on the optimization time *exponential* if it is $2^{\Omega(n^c)}$ and *polynomial* if it is $O(n^c)$ (for some constant $c > 0$). Note, that we denote the logarithm of some x to base 2 by $\log(x)$ and the logarithm to base e by $\ln(x)$. Important notations that are used throughout the thesis are summarized in Appendix A. Useful mathematical tools can be found in Appendix B. When referring to these tools during calculations, we sometimes use simply the number of the respective tool as abbreviation.

In the next two parts of this thesis we investigate two aspects of AIS for function optimization that fall into the described general schema. In Part II we consider different immune-inspired variation operators whereas in Part III a diversity mechanism called *aging* is examined. We introduce the concrete algorithms at the beginning of each part of the thesis.

Part II.

Analyses of Mutation in Artificial Immune Systems

3. Introduction

Variation is a crucial part of a randomized search heuristic since it enables the algorithm to explore the search space. An inappropriate choice of variation operators can lead to getting stuck in local optima or elsewhere. In the context of this part of the thesis we consider mutation as only variation operator.

Mutation in AIS is quite different from mutation in other randomized search heuristics, e. g., evolutionary algorithms. While in evolutionary algorithms generally moderate mutation probabilities are used, AIS incorporate mutations at high rate, also called *somatic hypermutations* (Bernardino and Barbosa 2009). Here, the term somatic means that mutations only effect the immune cells themselves but not the genetic material of the organism. Thus, mutations in the immune system and abilities learned by the adaptive immune system are not passed forward to offsprings. This is in contrast to mutations in the context of evolution.

Different types of immune-inspired hypermutation operators have been proposed in the literature, e. g., static, fitness-proportional hypermutation, inversely fitness-proportional hypermutation, and hypermacromutation (Cutello et al. 2004a) as well as somatic contiguous hypermutation (Kelsey and Timmis 2003). All these have there roots in different processes observed in the immune system. Probably the most prominent type of hypermutations is the use of inversely fitness-proportional mutation probabilities since this concept derives directly from the widely accepted clonal selection principle. Various implementations of this class of hypermutations exist. In the following, we briefly discuss common variants of hypermutation operators as well as related work from the field of evolutionary algorithms. Afterwards an overview over results in this thesis and the considered analytical framework are presented.

3.1. Related Work

One of the first clonal selection algorithms for optimization was CLONALG (de Castro and Von Zuben 2002b). It incorporates an inversely fitness-proportional mutation operator that uses the inverse of an exponential function to establish a relationship between the mutation probability and the normalized function value of the search point to be mutated. The immune network algorithm opt-aiNet (de Castro and Timmis 2002b) is quite similar to CLONALG. It uses inversely fitness-proportional mutation in a similar way but with a slightly different parametrization. Note, that mutation operators depending on the fitness of search points, i. e., the current state of the optimization process, are somehow similar to evolution strategies (Schwefel 1995) as they employ self-adapting mechanisms (de Castro and Von Zuben 2002b). Clearly, the obvious difference is that in fitness-based mutation the correlation between mutation probability and fitness of the search point is fixed, while it changes during the optimization process of evolution strategies by means of indirect selection. However, fitness-depending mutation is not a common concept in the field of evolutionary algorithms (Bernardino and Barbosa 2009). A noteworthy exception is the recently introduced variant of rank-based mutation (Cervantes and Stephens 2008) that was theoretically analyzed by Oliveto et al. (2009).

Both operators, CLONALG and opt-aiNet, exist in a continuous and discrete version. However, we only consider the discrete variants here. Thus, in the context of this thesis, i. e., pseudo-Boolean objective functions, the mutation probability determines the probability for each bit in a given bit string to be mutated. This is different when considering another well-known clonal selection algorithm, opt-IA (Cutello et al. 2004a). Here, a function called *mutation potential* determines the maximal number of mutation steps, i. e., flipping bits, during the execution of the mutation operator. Such mutation potentials exist in different flavors, e. g., static, fitness-proportional, and inversely fitness-proportional. Moreover, they can be restricted to certain regions of the bit string (hypermacromutation). In order to prevent getting stuck, additionally the *stop at the first constructive mutation* mechanism is employed. Here, after each single mutation step it is checked if the search point has improved and stopped if this is the case. A convergence analysis of an algorithmic framework incorporating opt-IA is available (Cutello et al. 2007c).

Another well-known variant of hypermutations used in AIS are somatic contiguous hypermutations which are inspired by the mutation mechanism found in B cell receptors. Here, mutation only effects a contiguous, randomly determined region of the search point, where each bit is flipped with a pre-defined fixed probability. The mutation operator was introduced as part of the B-Cell algorithm (Kelsey and Timmis 2003). A convergence analysis is presented by Clark et al. (2005) and Clark (2008) using a Markov chain model.

An extensive overview on applications and variants of these algorithms was presented by Brownlee (2007). We consider the mutation operators from CLONALG, opt-aiNet and the B-Cell algorithm in Chapter 4 and Chapter 5, respectively. The analysis of mutation potentials from opt-IA remains an interesting open problem.

In the field of evolutionary algorithms other theoretical investigations of mutation operators are available. For example, Jansen and Wegener (2000) showed that using an appropriate mutation probability is essential for the performance of the algorithms. In particular, they showed that the most recommended standard choice for the mutation probability ($1/n$) is far from optimal for some problems. Jansen and Sudholt (2010) and Doerr et al. (2007) investigated the use of asymmetric mutation operators. The interplay between mutation and selection is considered by Lehre and Yao (2011).

3.2. Contribution of this Thesis

In this part of the thesis we consider different aspects of mutation in artificial immune systems. We start with an analysis of the most common variant, namely inversely fitness-proportional mutation in Chapter 4. We investigate the role of parametrization and derive guidelines for setting embedded parameters appropriately. We compare different implementations of such mutation operators, both straightforward ideas and immune-inspired variants used in practical immune-inspired algorithms, i. e., CLONALG and opt-aiNet, on a well-known instructive example function. Thereby, the main focus is on the mutation incorporated into CLONALG. Besides the parametrization, we investigate the utility of larger collections of search points and thus, the role of the used method for normalizing function values. We present results of experiments to evaluate the practical relevance of our theoretical findings and investigate questions not proven in the analytical part.

In Chapter 5, we analyze contiguous hypermutations from the B-Cell algorithm in the same manner. We investigate benefits and drawbacks of this kind of mutation on several instructive example functions from the literature, among that the function used in the preceding chapter. We discuss serious limitations of this kind of mutation and point out in which situations it performs better than standard mutations in evolutionary computation. Again, we close the chapter by presenting experimental supplements.

In the last chapter of this part (Chapter 6), we consider a more general aspect of hypermutations. Keeping in mind results from the preceding analyses, we investigate negative effects of large mutation probabilities. We show that on the class of monotone functions, increasing the standard mutation probability from evolutionary algorithms by a constant factor can make a decisive difference, i. e., the optimization time increases from polynomial to exponential. Since in artificial immune systems even larger mutation probabilities are typical, this result is relevant for hypermutations and shows serious drawbacks of this kind of mutation operator.

3.3. A Simple Algorithmic Framework

In this part of the thesis, we focus on the analysis of the influence of different immune-inspired mutation operators and related variants of these operators on the performance of randomized search heuristics. This is most easily done if the considered operators can be studied as much in isolation as possible. We achieve this by omitting other possible features of an artificial immune system and implementing the different kinds of mutation in a minimal algorithmic framework.

The following randomized search heuristic uses a collection of search points of size μ. It works in rounds where in each round one new search point is generated as random variation of an existing search point and μ search points are selected for the next round. A more formal description of the algorithmic framework is given in Algorithm 3.1.

We use Algorithm 3.1 for maximization of an objective function $f\colon \{0,1\}^n \to \mathbb{R}$ or $f\colon \{0,1\}^n \to \mathbb{R}^+$, respectively, and implement the four modules in very simple ways. The initialization (line 1[1], Algorithm 3.2) is carried out uniformly at random. Variation creates one new search point y by means of mutation of a search point selected uniformly at random from the current collection of search points (line 2b, Algorithm 3.3). Here, the

[1]all line numbers from Algorithm 3.1

Algorithm 3.1 A Simple Algorithmic Framework.

```
1. Initialization
   Let $t := 0$ and initialize collection of search points $C_0$ of size $\mu$.
2. Repeat
   a) Let $t := t + 1$.
   b) Variation
      Generate new search point $y$.
   c) Selection for Replacement
      Decide if $y$ is to be inserted in $C_t$. Remove search point if needed.
   Until some termination condition is met.
```

mutation operators under consideration are plugged in. In the selection for replacement (line 2c, Algorithm 3.4) the new search point y is inserted if its function value is not worse than the worst function value of any of the current search points. In this case, one of the current worst search points is removed uniformly at random. We remark that depending on the context, we omit the index t of the current collection of search points if it is not needed.

Algorithm 3.2 Initialization.

```
1. Choose $x_1, \ldots, x_\mu \in \{0,1\}^n$ uniformly at random.
2. Set $C_0 := \{x_1, \ldots, x_\mu\}$.
```

Algorithm 3.3 Variation.

```
1. Select $x \in C_{t-1}$ uniformly at random.
2. Set $y := \text{mutate}(x)$.
```

Following the notation defined in Chapter 2, we denote the optimization time of Algorithm 3.1 using some mutation operator op by $T_{\text{op},f} = \min\{t \mid C_t \cap \text{OPT} \neq \emptyset\}$ where $\text{OPT} = \{x \in \{0,1\}^n \mid f(x) = \max\{f(x') \mid x' \in \{0,1\}^n\}\}$. Recall that $\mathrm{E}(T_{\text{op},f})$ is its expected value and $T_{\text{op},f,C}$ the optimization time if the algorithm is started in some specific initial collection of search points $C = \{x_1, \ldots, x_\mu\}$. The different variants of op are defined within the next chapters of the thesis.

We aim at comparing results for immune-inspired mutation operators with typical mutation operators from other randomized search heuristics. In evolutionary computation standard bit mutations (SBM) are most common (de Jong 2006). We define this type of mutation formally in Definition 3.5 and two well-known algorithms that are the result of using standard bit mutation within our algorithmic framework in Definition 3.1.

Definition 3.1 ((μ+1) EA, (1+1) EA)**.** *Plugging standard bit mutations (Algorithm 3.5) into the described framework yields the well-known* (μ+1) EA *(Witt 2006). Furthermore, setting $\mu = 1$, yields the* (1+1) EA *(Droste et al. 2002). Typically, $p(n) = 1/n$ holds.*

Algorithm 3.4 Selection for Replacement.

```
1. Choose $z \in C_{t-1}$ with minimal fitness uniformly at random,
       i. e., $z \in \{x \in C_{t-1} \mid f(x) = \min\{f(x') \mid x' \in C_{t-1}\}$
2. **If** $f(y) \geq f(z)$ **then**
3.     Set $C_t := (C_{t-1} \cup \{y\}) \setminus \{z\}$.
4. **Else**
5.     Set $C_t := C_{t-1}$.
```

Algorithm 3.5 Standard bit mutation (SBM).

```
FUNCTION **mutate($x$):**
1. Let $y := x$.
1. Independently for each $i \in \{0, 1, \dots, n-1\}$
2.     With probability $p(n)$ set $y[i] := 1 - y[i]$.
```

We remark that the proposed algorithmic framework is rather general. In particular, it includes among others randomized local search (Michiels et al. 2007) and tabu search (Glover and Laguna 1997). Thus, available results for these algorithms, see e. g., (Auger and Doerr 2011; Neumann and Witt 2010) may be used for comparisons, contributing to the understanding of similarities and differences of the different approaches.

4. Inversely Fitness-Proportional Mutation Rates

We first investigate the probably most common concept of hypermutation operators in the field of AIS, namely inversely fitness-proportional mutation probabilities. This concept has its root in the affinity maturation process, inherent in the clonal selection principle. Such mutation operators are claimed to be especially appropriate for multi-modal function optimization since each candidate solution has an independent, adaptive mutation probability (de Castro and Von Zuben 2002b). Search points in 'better' regions of the search space are only subject to small mutation probabilities while for search points located far away from optimal regions larger mutation probabilities are used.

We investigate several variants of inversely fitness-proportional mutation probabilities using the framework described in the previous section. For most of our results we consider Algorithm 3.1 using the mutation operators under consideration on the well-known example function OneMax, which counts the number of 1-bits in a bit string and can formally be defined as follows.

Definition 4.1. *For $n \in \mathbb{N}$ and $x \in \{0,1\}^n$, the function* OneMax: $\{0,1\}^n \to \mathbb{R}$ *is defined by*

$$\text{OneMax}(x) = \sum_{i=0}^{n-1} x[i].$$

It is known that the (1+1) EA solves OneMax in expected time $\Theta(n \log n)$ (Droste et al. 2002). As a preliminary step, we start with two very simple and straightforward implementations of inversely fitness-proportional mutation operators. Afterwards, we consider two immune-inspired variants introduced within the optimization algorithm CLONALG (de Castro and Von Zuben 2002b) and opt-aiNet (de Castro and Timmis 2002b). For the immune-inspired operators, our main focus lies on the influence of parameters embedded in the operators. Finally, all our theoretical findings are accompanied by empirical results.

The results in this chapter are mainly based on the work done in Zarges (2008, 2009). However, Sections 4.1 and 4.4 were not published before. The empirical analysis in Section 4.5 was significantly extended in comparison to Zarges (2008). Moreover, the proofs in Sections 4.2 and 4.3.1 were partly restructured.

4.1. A Straightforward Implementation

We start our investigations with a straightforward implementation of the mathematical term of inverse proportionality. This mutation probability was proven to be the optimal adaptive mutation probability for the example function LeadingOnes (see Definition 5.8) that counts the number of leading 1-bits in a bit string (Böttcher et al. 2010). We define the resulting mutation operator in Algorithm 4.1.

Algorithm 4.1 Adaptive mutation probability p_{simple} (Böttcher et al. 2010).

```
FUNCTION mutate(x):
1.  Let $v := f(x)$ and $y := x$.
2.  Independently for each $i \in \{0, 1, \ldots, n-1\}$
3.      With probability $p_{\text{simple}}(v) := 1/(v+1)$ set $y[i] := 1 - y[i]$.
```

It is easy to see that plugging this mutation operator into our algorithmic framework (Algorithm 3.1) yields polynomial optimization time on OneMax. Moreover, one can recognize that the mutation probability is not too different from a standard bit mutation where each bit is flipped with some probability c/n ($c > 0$ constant). We formalize this in the following theorem.

Theorem 4.2. *Let $\mu = 1$. The expected optimization time of Algorithm 3.1 using the mutation operator from Algorithm 4.1 on* OneMax *is*

$$E\left(T_{p_{simple},\textsc{OneMax}}\right) = \Theta(n \log n).$$

Proof. We first observe, that for all $x \in \{0,1\}^n$ with $\textsc{OneMax}(x) \geq cn = \Omega(n)$ ($c > 0$ constant), we have a mutation probability of $p_{\text{simple},\textsc{OneMax}} \leq 1/(cn+1) = O(1/n)$. Moreover, $p_{\text{simple},\textsc{OneMax}} \geq 1/(n+1)$. Thus, for these bit strings $p_{\text{simple},\textsc{OneMax}} = \Theta(1/n)$.

It is known that the optimization time of Algorithm 3.1 with $\mu = 1$ using mutation probability $\Theta(1/n)$ is $\Theta(n \log n)$ (Droste et al. 2002), but the proof of this result assumes a fixed mutation probability throughout the whole optimization process, independently of the current function value. However, the bound can be carried over easily by adapting the fitness-level arguments incorporated. Given a search point with $\textsc{OneMax}(x) = i$, the probability to increase the function value by at least one can be bounded below by

$$\binom{n-i}{1} \cdot \frac{1}{i+1} \cdot \left(1 - \frac{1}{i+1}\right)^{n-1} \overset{\text{(B.8)}}{\geq} \frac{n-i}{i+1} \cdot \left(\frac{1}{e}\right)^{\frac{n-1}{i}}$$

Since the waiting times are geometrically distributed (Lemma B.14), this yields for $i \geq cn = \Omega(n)$

$$\sum_{i=cn}^{n-1} \frac{i+1}{n-i} \cdot e^{\frac{n-1}{i}} \leq e^{\frac{1}{c}} \cdot n \cdot \sum_{i=cn}^{n-1} \frac{1}{n-i} = e^{\frac{1}{c}} \cdot n \cdot \sum_{i=1}^{(1-c)n} \frac{1}{i} \overset{\text{(B.6)}}{=} O(n \log n)$$

for the expected optimization time. Note, that due to Chernoff bounds (see Lemma B.21), we have $\textsc{OneMax}(x_0) \geq dn$ ($d < 1/2$ constant) with probability $1 - 2^{-\Omega(n)}$ for the initial search point x_0. But we still need to investigate the mutation probability if $\textsc{OneMax}(x) = o(n)$

Let $z = n - \textsc{OneMax}(x)$ denote the current number of 0-bits. The probability that the number of 0-bits decreases by some amount i can be bounded below by the probability that exactly i 0-bits and none of the 1-bits flips. Using the definition of the binomial distribution (see Lemma B.13) in eq. (4.1) and Lemma B.8 in eq. (4.2), we get

$$\begin{aligned}
&\sum_{i=1}^{z} \binom{z}{i} \cdot \left(\frac{1}{n-z+1}\right)^{i} \cdot \left(1 - \frac{1}{n-z+1}\right)^{n-i} \\
&= \left(1 - \frac{1}{n-z+1}\right)^{n-z} \cdot \sum_{i=1}^{z} \binom{z}{i} \cdot \left(\frac{1}{n-z+1}\right)^{i} \cdot \left(1 - \frac{1}{n-z+1}\right)^{z-i} \\
&= \left(1 - \frac{1}{n-z+1}\right)^{n-z} \cdot \left(\sum_{i=0}^{z} \binom{z}{i} \cdot \left(\frac{1}{n-z+1}\right)^{i} \cdot \left(1 - \frac{1}{n-z+1}\right)^{z-i} \right. \\
&\qquad \left. - \binom{z}{0} \cdot \left(\frac{1}{n-z+1}\right)^{0} \cdot \left(1 - \frac{1}{n-z+1}\right)^{z-0}\right) \\
&= \left(1 - \frac{1}{n-z+1}\right)^{n-z} \cdot \left(1 - \left(1 - \frac{1}{n-z+1}\right)^{z}\right) \qquad (4.1) \\
&= \left(1 - \frac{1}{n-z+1}\right)^{n-z} - \left(1 - \frac{1}{n-z+1}\right)^{n} \\
&\geq \left(\frac{1}{e}\right) - \left(\frac{1}{e}\right)^{\frac{n}{n-z+1}} \qquad (4.2) \\
&= \Theta(1) \qquad \text{for } \textsc{OneMax}(x) = n - z = o(n).
\end{aligned}$$

Altogether, this yields

$$\mathrm{E}\left(T_{p_{\text{simple}}, \textsc{OneMax}}\right) = O(n \log n) + o(n) = O(n \log n).$$

For the lower bound and $\textsc{OneMax}(x) = \Omega(n)$, we can again use the argumentation by Droste et al. (2002). Assume w. l. o. g. that n is even. With probability at least $1/2 - 2^{-\Omega(n)}$, we have $n/2 \leq |x_0|_0 \leq 2n/3$ 0-bits in the initial search point x_0. The probability that a bit does not flip in t iterations becomes minimal for the maximal mutation probability. Therefore, let c/n for some constant $c > 1$ be the maximal mutation probability during the considered process. Then the probability that at least one of $n/2$ bits never flips in t steps is at least $1 - \left(1 - (1 - c/n)^t\right)^{n/2}$. We consider $t = c^{-1}(n - c) \ln n$

iterations and get

$$\begin{aligned}
\mathrm{E}\left(T_{p_{\text{simple}},\textsc{OneMax}}\right) &= \sum_{t=1}^{\infty} t \cdot \mathrm{Prob}\left(T_{p_{\text{simple}},\textsc{OneMax}} = t\right)\\
&= \sum_{t=1}^{\infty} \mathrm{Prob}\left(T_{p_{\text{simple}},\textsc{OneMax}} \geq t\right)\\
&\geq \left(\frac{1}{2} - 2^{-\Omega(n)}\right) \cdot \sum_{t=1}^{\infty} \left(1 - \left(1 - \left(1 - \frac{c}{n}\right)^{t}\right)^{\frac{n}{2}}\right)\\
&\geq \left(\frac{1}{2} - 2^{-\Omega(n)}\right) \cdot \left(\frac{n}{c} - 1\right) \ln n \left(1 - \left(1 - \left(1 - \frac{c}{n}\right)^{\left(\frac{n}{c}-1\right)\ln n}\right)^{\frac{n}{2}}\right)\\
&\geq \left(\frac{1}{2} - 2^{-\Omega(n)}\right) \cdot \left(\frac{n}{c} - 1\right) \ln n \left(1 - \left(1 - e^{-\ln n}\right)^{\frac{n}{2}}\right)\\
&\geq \left(\frac{1}{2} - 2^{-\Omega(n)}\right) \cdot \left(\frac{n}{c} - 1\right) \ln n \left(1 - e^{-\frac{1}{2}}\right)\\
&= \Omega(n \log n). \qquad \square
\end{aligned}$$

4.2. Hamming Distance Based Mutation Rates

As done by Zarges (2008), we consider another simple, fitness-proportional mutation probability that is not immune-based. Here, the mutation probability depends on the minimal Hamming distance of the current search point $x \in \{0,1\}^n$ and an optimal search point of the considered function. Clearly, for OneMax the Hamming distance to the global optimum is equivalent to the difference of the respective function values. We define the resulting mutation operator in Algorithm 4.2.

Algorithm 4.2 Hamming distance based mutation probability p_{Hamming} (Zarges 2008).

```
FUNCTION mutate($x$):
1. Let $v := \min\{H(x,z) \mid z \in \mathrm{OPT}\}$ with
       $\mathrm{OPT} = \{x \in \{0,1\}^n \mid f(x) = \max\{f(x') \mid x' \in \{0,1\}^n\}\}$.
2. Let $y := x$.
3. Independently for each $i \in \{0,1,\ldots,n-1\}$
4.     With probability $p_{\text{Hamming}}(v) := v/n$ set $y[i] := 1 - y[i]$.
```

Since the expected number of flipping bits for this operator is v, such a mutation probability is optimal if the objective is to maximize the probability to find the global optimum of OneMax in a single mutation step. However, we show that this is not helpful with respect to the optimization time. Note, that for a convincing lower bound on the optimization time it is insufficient to prove a large lower bound on the expected optimization time. Large expected values may be misleading owing their magnitude to unlikely events leading to exceedingly long runs. It is therefore much more informative to have a lower bound on the probability that a run takes long. This is exactly the kind of statement we make in the following theorem. We first prove that with overwhelming probability, the optimum of OneMax is not found within an exponential number of iterations. We remark, that for search point x with $\text{OneMax}(x) \leq n/2$ we have $p_{\text{Hamming}} \geq 1/2$.

Theorem 4.3 (Zarges (2008))**.** *Let $\mu = 1$. The optimization time of Algorithm 3.1 using the mutation operator from Algorithm 4.2 on* OneMax *is*

$$T_{p_{Hamming},\text{OneMax}} \geq e^{c \cdot n}$$

($c > 0$ constant, sufficiently small) with probability at least $1 - e^{-\Omega(n)}$.

Proof. Let z denote the number of 0-bits in the current search points, M_0 the number of flipping 0-bits and M_1 the number of flipping 1-bits, respectively. Then, it is easy to see that

$$\mathrm{E}\left(M_0\right) = \frac{z^2}{n} \qquad \text{and} \qquad \mathrm{E}\left(M_1\right) = z - \frac{z^2}{n}$$

holds. Moreover, due to linearity of expectation (Lemma B.12) the expected progress in an iteration equals

$$\mathrm{E}\left(M_0 - M_1\right) = \frac{2z^2}{n} - z.$$

By Chernoff bounds (Lemma B.21), we have $\text{Prob}\left(n/3 \leq \text{OneMax}(x_0) \leq 5n/6\right) = 1 - e^{-\Omega(n)}$ for the initial search point x_0. In the following, we show that as long as $\text{OneMax}(x) \leq 5n/6$ holds, we have $\text{Prob}\left(\text{OneMax}(y) > 5n/6\right) = e^{-\Omega(n)}$. Let δ be some constant with $0 < \delta < 1/6$. We distinguish two cases. The main ideas of the proofs are visualized in Figure 4.1 where the red line corresponds to the first case and the green one to the second case. Note, that $\mathrm{E}\left(M_0\right) = \Omega(n)$ and $\mathrm{E}\left(M_1\right) = \Omega(n)$ hold in both cases.

1. $(1/2 + \delta) \cdot n \leq \text{OneMax}(x) \leq 5n/6$:

   In the first case, we show a large negative drift that hinders the algorithm to increase the function value, in particular beyond $5n/6$. Again due to Chernoff bounds, we have

$$\text{Prob}\left(M_0 > (1+\delta) \cdot \mathrm{E}\left(M_0\right)\right) \leq e^{-\frac{\delta^2 \mathrm{E}(M_0)}{3}} = e^{-\Omega(n)} \text{ and}$$
$$\text{Prob}\left(M_1 < (1-\delta) \cdot \mathrm{E}\left(M_1\right)\right) \leq e^{-\frac{\delta^2 \mathrm{E}(M_1)}{2}} = e^{-\Omega(n)}.$$

For $z = n - \textsc{OneMax}(x) \leq (1/2 - \delta) \cdot n$, we can conclude that

$$\begin{aligned}
M_0 - M_1 &\leq (1+\delta) \cdot \mathrm{E}\left(M_0\right) - (1-\delta) \cdot \mathrm{E}\left(M_1\right) \\
&= (1+\delta) \cdot \frac{z^2}{n} - (1-\delta) \cdot \left(z - \frac{z^2}{n}\right) \\
&= \frac{z^2}{n} + \delta \cdot \frac{z^2}{n} - z + \frac{z^2}{n} + \delta z - \delta \cdot \frac{z^2}{n} \\
&= z \cdot \left(\frac{2z}{n} + \delta - 1\right) \\
&\leq z \cdot \left(\frac{2 \cdot \left(\frac{1}{2} - \delta\right) \cdot n}{n} + \delta - 1\right) \\
&= -\delta \cdot z < 0
\end{aligned}$$

holds with probability $1 - e^{-\Omega(n)}$. Altogether, this yields

$$\begin{aligned}
&\mathrm{Prob}\left(\textsc{OneMax}(y) > \textsc{OneMax}(x)\right) = \mathrm{Prob}\left(M_0 > M_1\right) \\
\leq &\mathrm{Prob}\left(M_0 > (1+\delta) \cdot E(M_0)\right) \cdot \mathrm{Prob}\left(M_1 < (1-\delta) \cdot E(M_1)\right) = e^{-\Omega(n)}
\end{aligned}$$

for the first case.

2. $n/3 \leq \textsc{OneMax}(x) < (1/2 + \delta) \cdot n$:

   In the second case, we show that with overwhelming probability we either stay in the second case or 'jump' to the region of the first case. Note that the expected progress $\mathrm{E}\left(M_0 - M_1\right)$ is maximal for $\textsc{OneMax}(x) = n/3$ in the considered region of the search space. Let z' be the number of 0-bits after mutating a search point with $(1/2 - \delta) \cdot n < z < 2n/3$ 0-bits. Due to the above calculations, we get the following lower bound on z'.

$$\begin{aligned}
z' &\geq z - z \cdot \left(\frac{2z}{n} + \delta - 1\right) = z \cdot \left(2 - \frac{2z}{n} - \delta\right) \\
&> \left(\left(\frac{1}{2} - \delta\right) \cdot n\right) \cdot \left(2 - \frac{4}{3} - \delta\right) = n \cdot \left(\frac{1}{2} - \delta\right) \cdot \left(\frac{2}{3} - \delta\right) \\
&> n \cdot \left(\frac{1}{2} - \frac{1}{6}\right) \cdot \left(\frac{2}{3} - \frac{1}{6}\right) = \frac{n}{6}
\end{aligned}$$

   Since $z' > n/6$ implies $\textsc{OneMax}(y) < 5n/6$, we conclude that with probability $1 - 2^{-\Omega(n)}$ we do not improve the function value beyond $5n/6$.

We have shown that $\mathrm{Prob}\left(\textsc{OneMax}(y) > 5n/6\right) = e^{-\Omega(n)}$ holds if $n/3 \leq \textsc{OneMax}(x_0) \leq 5n/6$. Together this shows that with probability $1 - e^{-\Omega(n)}$ even in $e^{c \cdot n}$ iterations the global optimum of $\textsc{OneMax}$ is not found ($c > 0$ constant, sufficiently small). □

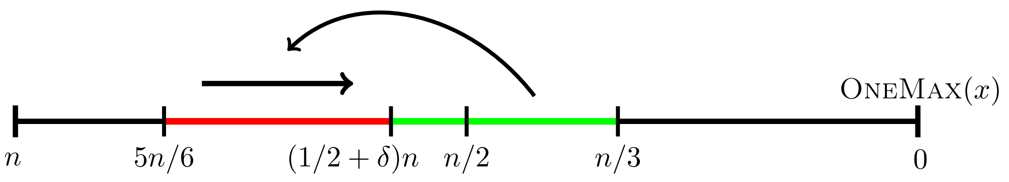


Figure 4.1.: Visualization of the drift in the proof of Theorem 4.3.

We have seen that the Hamming distance based mutation probability yields an exponential lower bound on the optimization time if the collection of search points in Algorithm 3.1 is initialized uniformly at random. Assume, the initial search point is located in some specific region of the search space. The next theorem proves that in this case, we can achieve a polynomial optimization time with high probability.

Theorem 4.4 (Zarges (2008))**.** *Let $\mu = 1$ and x_0 be the initial search point. Assume* ONEMAX$(x_0) \leq c \ln n$ *or* ONEMAX$(x_0) \geq n - c \ln n$ *for some constant $c > 0$. The optimization time of Algorithm 3.1 using the mutation operator from Algorithm 4.2 and on* ONEMAX *is*

$$T_{p_{Hamming},\textsc{OneMax},x_0} = O\big(n^{c+1} \log n\big)$$

with probability $1 - n^{-\omega(1)}$.

Proof. Let $M = M_0 + M_1$ denote the total number of flipping bits in an iteration, where M_0 is the number of flipping 0-bits and M_1 the number of flipping 1-bits. Moreover, let $\overline{M} = n - M$. We start with the case ONEMAX$(x_0) = O(\log n)$ and show that after one iteration we are, with probability $1 - n^{-\omega(1)}$, in the second case, i. e., ONEMAX(x_0) $= n - O(\log n)$. Afterwards, we show that in this situation, we find the global optimum of ONEMAX in expected polynomial time.

With ONEMAX$(x_0) = O(\log n)$, we have $p_{\text{Hamming}}(v) = (n - O(\log n))/n$ and thus, $\mathrm{E}\left(\overline{M}\right) = O(\log n)$. By Chernoff bounds (see Lemma B.21), the probability that $\alpha(n)$ $= \omega(\log n)$ bits do not flip can be bounded above as follows.

$$\begin{aligned}\text{Prob}\left(\overline{M} \geq \alpha(n)\right) &= \text{Prob}\left(\overline{M} \geq \left(1 + \left(\frac{\alpha(n)}{c} - 1\right)\right) \cdot c\right) \\ &< \left(\frac{e^{\frac{\alpha(n)}{c} - 1}}{\left(\frac{\alpha(n)}{c}\right)^{\frac{\alpha(n)}{c}}}\right)^{c} \leq \left(\frac{ce}{\alpha(n)}\right)^{\alpha(n)} = 2^{-\omega(\log n)} = n^{-\omega(1)}\end{aligned}$$

Thus, we have $\overline{M} = O(\log n)$ with probability at least $1 - n^{-\omega(1)}$. We pessimistically assume that all 1-bits flip, i. e., $M_1 = O(\log n)$. However, since with probability at least $1 - n^{-\omega(1)}$ only $O(\log n)$ 0-bits do not flip, we have ONEMAX$(y) = n - O(\log n)$ with probability $1 - n^{-\omega(1)}$.

For the case $\textsc{OneMax}(x_0) = n - O(\log n)$, say $n - c\ln n$ for some $c = O(1)$, we first estimate the success probability in one iteration, i. e., the probability to flip more 0-bits than 1-bits. For the sake of readability, we define $k = n - \textsc{OneMax}(x)$ for some search point x. Using Lemma B.16 for eq. (4.3) and Lemma B.8 as well as the expectation of the binomial distribution (see Lemma B.13) in eq. (4.4), we get

$$\begin{aligned}
&\operatorname{Prob}(M_0 > M_1)\\
&= \sum_{i=1}^{k}\sum_{j=0}^{i-1}\binom{k}{i}\cdot\binom{n-k}{j}\cdot\left(\frac{k}{n}\right)^{i+j}\cdot\left(1-\frac{k}{n}\right)^{n-i-j}\\
&= \sum_{i=1}^{k}\binom{k}{i}\cdot\left(\frac{k}{n}\right)^{i}\cdot\left(1-\frac{k}{n}\right)^{k-i}\cdot\left(\sum_{j=0}^{i-1}\binom{n-k}{j}\cdot\left(\frac{k}{n}\right)^{j}\cdot\left(1-\frac{k}{n}\right)^{n-k-j}\right)\\
&\geq \sum_{i=1}^{k}\binom{k}{i}\cdot\left(\frac{k}{n}\right)^{i}\cdot\left(1-\frac{k}{n}\right)^{k-i}\cdot\left(\sum_{j=0}^{i-1}\left(\frac{n-k}{k}\right)^{j}\cdot\left(\frac{k}{n}\right)^{j}\cdot\left(1-\frac{k}{n}\right)^{n-k-j}\right) &(4.3)\\
&= \sum_{i=1}^{k}\binom{k}{i}\cdot\left(\frac{k}{n}\right)^{i}\cdot\left(1-\frac{k}{n}\right)^{k-i}\cdot\left(\sum_{j=0}^{i-1}\left(1-\frac{k}{n}\right)^{n-k}\right)\\
&= \left(1-\frac{k}{n}\right)^{n-k}\cdot\sum_{i=1}^{k} i\cdot\binom{k}{i}\cdot\left(\frac{k}{n}\right)^{i}\cdot\left(1-\frac{k}{n}\right)^{k-i}\\
&\geq \left(\frac{1}{e}\right)^{k}\cdot\frac{k^2}{n}. &(4.4)
\end{aligned}$$

Remember, that $k = n - \textsc{OneMax}(x)$ and thus, with $\textsc{OneMax}(x_0) = n - c\ln n = n - O(\log n)$, we have $k \in \{1, \ldots, c\ln n\}$. This yields an expected optimization time of at most

$$\mathrm{E}\left(T_{p_{\text{Hamming}},\textsc{OneMax},x_0}\right) \leq \sum_{k=1}^{c\ln n}\frac{e^k n}{k^2} \leq c\ln n\cdot e^{c\ln n}\cdot n = O\left(n^{c+1}\log n\right)$$

Altogether, for $\textsc{OneMax}(x_0) = n - O(\log n)$ or $\textsc{OneMax}(x_0) = O(\log n)$ we find the global optimum in a polynomial number of iterations with probability $1 - n^{-\omega(1)}$. $\square$

From the proof of Theorem 4.4, we can immediately conclude that for one of the considered cases, namely if the initial search point is located 'near' to the optimum, we also have an expected polynomial optimization time.

Corollary 4.5. *Let $\mu = 1$ and x_0 be the initial search point. Assume $\textsc{OneMax}(x_0) \geq n - c\ln n$ for some constant $c > 0$. The expected optimization time of Algorithm 3.1 using the mutation operator from Algorithm 4.2 on* $\textsc{OneMax}$ *is*

$$E\left(T_{p_{Hamming},\textsc{OneMax},x_0}\right) = O\left(n^{c+1}\log n\right).$$

Proof. As seen in the proof of Theorem 4.4 the failure probability $1 - n^{-\omega(1)}$ derives from the case $\text{OneMax}(x_0) \leq c \ln n$. Once we have $\text{OneMax}(x_0) \geq n - c \ln n$, the function value never decreases due to the elitist selection scheme. This yields the claim on the expected optimization time for the case where we initialize 'near' to the optimum. □

4.3. The Mutation Operator from CLONALG

As a first example for an immune-based mutation operator, we consider the one proposed for the well-known clonal selection algorithm CLONALG (de Castro and Von Zuben 2002b). Originally, this mutation operator was introduced in the context of continuous optimization, but since we concentrate on discrete optimization we only consider its discrete variant here.

The operator uses the inverse of an exponential function to establish a relationship between the mutation probability and the normalized function value of the search point to be mutated. A parameter ρ controls the smoothness of this inverse exponential function (often called *decay* parameter). The mutation operator can formally be defined as described in Algorithm 4.3.

Algorithm 4.3 CLONALG mutation probability p_{CLONALG} (de Castro and Von Zuben 2002b).

```
FUNCTION mutate(x):
1. Let $v := \text{normalize}(f(x)) \in [0,1]$ and $y := x$.
2. Independently for each $i \in \{0, 1, \ldots, n-1\}$
3.     With probability $p_{\text{CLONALG}}(v) := e^{-\rho \cdot v}$ set $y[i] := 1 - y[i]$.
```

When using the above mutation operator one has to decide about an appropriate parameter setting for ρ and a normalization method. In the following, we consider these two aspects. Like in the sections before, we first investigate the performance of the operator in the very simple framework where the collection of search points only contains one search point. Since we mainly show negative results for this case, we extend the framework by considering a larger collection of search points and as a consequence another normalization method which yields better optimization times.

4.3.1. The Role of the Decay Parameter ρ

We first investigate the influence of the decay parameter ρ within a very simple algorithmic framework where we set $\mu = 1$ just like in the previous two sections. As a consequence we need to discuss the normalization method used. Usually, in case of a maximization problem, the function value is normalized by dividing the fitness of the considered search point by the fitness of the best current search point. Since our collection of search points only consists of a single search point, we use the optimal value of the considered objective function instead. Clearly, in case of OneMax this is n. We define the normalization method formally in Algorithm 4.4 and obtain

$$p_{\mathrm{CLONALG}}(\textsc{OneMax}(x)) = e^{-\rho\cdot\frac{\textsc{OneMax}(x)}{n}} \tag{4.5}$$

for the rest of this section.

Algorithm 4.4 Normalizing by means of the optimal function value f_{OPT}.

```
FUNCTION normalize(v):
1. Return $v/f_{\mathrm{OPT}}$.
```

We remark, that in real applications the optimal value of the considered function is generally not known and thus, it is difficult to apply this concept in practice. However, in real applications one could alternatively use an upper bound for the optimal value. Note, that using an upper bound for the optimal value leads to larger mutation probabilities, whereas the use of the fitness of the best current search point in a collection of search points yields smaller mutation probabilities. We come back to this point later in Section 4.3.2 when discussing the use of a larger collection of search points.

In the following, we consider different settings of the decay parameter ρ and show that an appropriate choice is essential for the performance of the mutation operator. We first analyze the two extreme cases $\rho = O(1)$ and $\rho = \Omega(n)$ and prove that with a probability close to 1 for both parameter settings the considered algorithm will not find the optimum of OneMax within an exponential number of iterations. Afterwards, we investigate how a carefully chosen intermediate value, i. e., $\rho = \ln n$, influences the optimization time.

Setting $\rho = O(1)$

Setting ρ to some constant value $d > 0$, the mutation probability from eq. (4.5) becomes

$$p_{\mathrm{CLONALG}}(\textsc{OneMax}(x)) = e^{-d\cdot\frac{\textsc{OneMax}(x)}{n}} = \Theta(1).$$

In the following we show that this mutation probability is much too large to be effective since it causes a large negative drift in relevant regions of the search space.

Theorem 4.6 (Zarges (2008))**.** *Let $\mu = 1$ and $\rho = O(1)$. The optimization time of Algorithm 3.1 using the mutation operator from Algorithm 4.3 and the normalization method from Algorithm 4.4 is*

$$T_{p_{CLONALG},\textsc{OneMax}} \geq e^{c\cdot n}$$

($c > 0$ constant, sufficiently small) with probability at least $1 - e^{-\Omega(n)}$.

Proof. The proof follows the line of thought of Theorem 4.3. Again, let z denote the number of 0-bits in the current search point, M_0 the number of flipping 0-bits and M_1 the number of flipping 1-bits, respectively. Then, we have

$$\begin{aligned}
\mathrm{E}(M_0) &= z \cdot e^{-\frac{d\cdot(n-z)}{n}}\\
\mathrm{E}(M_1) &= (n-z) \cdot e^{-\frac{d\cdot(n-z)}{n}}\\
\mathrm{E}(M_0 - M_1) &= (2z-n) \cdot e^{-\frac{d\cdot(n-z)}{n}}.
\end{aligned}$$

By Chernoff bounds (see Lemma B.21), we have $\text{Prob}\left(n/3 \leq \text{ONEMAX}(x_0) \leq 11n/12\right) = 1 - e^{-\Omega(n)}$ for the initial search point x_0. Then, analogously to Theorem 4.3, we show $\text{Prob}\left(\text{ONEMAX}(y) > 11n/12\right) = e^{-\Omega(n)}$ as long as $\text{ONEMAX}(x) \leq 11n/12$. Let δ be some constant with $0 < \delta < 1/12$ and distinguish the following two cases. Note, that again $\text{E}\left(M_0\right) = \Omega(n)$ and $\text{E}\left(M_1\right) = \Omega(n)$ hold in both cases. Despite the similarity of the calculations here and in the proof of Theorem 4.3, we repeat the complete calculations, since the concrete constants differ significantly from those in the previous proof.

1. $(1/2 + \delta) \cdot n \leq \text{ONEMAX}(x) \leq 11n/12$:

   For this case, we show a large negative drift that hinders the algorithm to increase the function value beyond $11n/12$. Again due to Chernoff bounds, we have $\text{Prob}\left(M_0 > (1+\delta) \cdot \text{E}\left(M_0\right)\right) = e^{-\Omega(n)}$ and $\text{Prob}\left(M_1 < (1-\delta) \cdot \text{E}\left(M_1\right)\right) = e^{-\Omega(n)}$. For $z = n - \text{ONEMAX}(x) \leq (1/2 - \delta) \cdot n$, we conclude that

$$\begin{aligned} M_0 - M_1 &\leq (1+\delta) \cdot \text{E}\left(M_0\right) - (1-\delta) \cdot \text{E}\left(M_1\right) \\ &= (1+\delta) \cdot z \cdot e^{-\frac{d \cdot (n-z)}{n}} - (1-\delta) \cdot (n-z) \cdot e^{-\frac{d \cdot (n-z)}{n}} \\ &= e^{-\frac{d \cdot (n-z)}{n}} \cdot \left((1+\delta) \cdot z - (1-\delta) \cdot (n-z)\right) \\ &\leq e^{-\frac{d \cdot (n-z)}{n}} \cdot \left((1+\delta) \cdot \left(\frac{1}{2} - \delta\right) \cdot n - (1-\delta) \cdot \left(\frac{1}{2} + \delta\right) \cdot n\right) \\ &= e^{-\frac{d \cdot (n-z)}{n}} \cdot n \cdot \left(\frac{1}{2} - \delta + \frac{\delta}{2} - \delta^2 - \left(\frac{1}{2} + \delta - \frac{\delta}{2} - \delta^2\right)\right) \\ &\leq -\delta \cdot n < 0 \end{aligned}$$

   holds with probability $1 - e^{-\Omega(n)}$. This yields

$$\begin{aligned} &\text{Prob}\left(\text{ONEMAX}(y) > \text{ONEMAX}(x)\right) = \text{Prob}\left(M_0 > M_1\right) \\ &\leq \text{Prob}\left(M_0 > (1+\delta) \cdot E(M_0)\right) \cdot \text{Prob}\left(M_1 < (1-\delta) \cdot E(M_1)\right) = e^{-\Omega(n)} \end{aligned}$$

   for the first case.

2. $n/3 \leq \text{ONEMAX}(x) < (1/2 + \delta) \cdot n$:

   In the second case, we show that with overwhelming probability we either stay in the second case or 'jump' to the region of the first case. Note, that the expected progress $\text{E}\left(M_0 - M_1\right)$ is again maximal for $\text{ONEMAX}(x) = n/3$. Let z' be the number of 0-bits after mutating a search point with $(1/2 - \delta) \cdot n < z < 2n/3$ 0-bits.

Due to the above calculations, we get the following lower bound on z'.

$$\begin{aligned}
z' \geq & z - e^{-\frac{d\cdot(n-z)}{n}} \cdot ((1+\delta)\cdot z - (1-\delta)\cdot(n-z)) \\
> & \left(\frac{1}{2}-\delta\right)\cdot n - e^{-\frac{d}{3}} \cdot \left((1+\delta)\cdot\frac{2n}{3} - (1-\delta)\cdot\frac{n}{3}\right) \\
= & n\cdot\left(\left(\frac{1}{2}-\delta\right) - e^{-\frac{d}{3}}\cdot\left(\frac{1}{3}+\delta\right)\right) \\
> & n\cdot\left(\left(\frac{1}{2}-\frac{1}{12}\right) - e^{-\frac{d}{3}}\cdot\left(\frac{1}{3}+\frac{1}{12}\right)\right) \\
= & \left(\frac{5}{12} - e^{-\frac{d}{3}}\cdot\frac{5}{12}\right)\cdot n > \frac{n}{12}
\end{aligned}$$

Since $z' > n/12$ implies $\textsc{OneMax}(y) < 11n/12$, we conclude that with probability $1-2^{-\Omega(n)}$ the function value does not increase beyond $11n/12$.

We have shown that $\text{Prob}\,(\textsc{OneMax}(y) > 11n/12) = e^{-\Omega(n)}$ holds if $n/3 \leq \textsc{OneMax}(x_0) \leq 11n/12$. Together this shows that with probability $1-e^{-\Omega(n)}$ even in $e^{c\cdot n}$ iterations the global optimum of OneMax is not found ($c > 0$ constant, sufficiently small). □

We remark, that for $\rho = O(1)$ we cannot show a polynomial upper bound on the optimization time for specific regions of the search space (compare Theorem 4.4), as, e. g., for $\textsc{OneMax}(x_0) = n - O(\log n)$ or even $\textsc{OneMax}(x_0) = n - O(1)$ the expected progress equals $O(\log n) - \Omega(n)$ and $O(1) - \Omega(n)$, respectively. We conclude that the parameter ρ should depend on the dimension n of the search space.

Setting $\rho = \Omega(n)$

For the other extreme case, we see that the mutation probability becomes much too small to be effective. To be more precise, we have

$$p_{\text{CLONALG}}(\textsc{OneMax}(x)) = e^{-\Omega(n)\cdot\frac{\textsc{OneMax}(x)}{n}} = e^{-\Omega(\textsc{OneMax}(x))},$$

yielding an expected exponential waiting time for a mutation actually changing a bit once the function value has increased to $\textsc{OneMax}(x) = \Omega(n)$. This yields an expected exponential optimization in the considered framework.

Theorem 4.7 (Zarges (2008))**.** *Let $\mu = 1$ and $\rho = \Omega(n)$. The expected optimization time of Algorithm 3.1 using the mutation operator from Algorithm 4.3 and the normalization method from Algorithm 4.4 on* OneMax *is*

$$E\left(T_{p_{clonalg},\textsc{OneMax}}\right) = e^{\Omega(n)}.$$

Proof. By Chernoff bounds (see Lemma B.21), we have $\text{Prob}\left(\frac{n}{3} \leq \textsc{OneMax}(x_0) \leq \frac{2n}{3}\right) = 1 - e^{-\Omega(n)}$ for the initial search point x_0. In this case, the mutation probability equals

$e^{-\Omega(n)}$ and the expected number of flipping bits in an iteration is n/e^{cn} for some constant $c > 0$. This yields

$$\mathrm{E}\left(T_{p_{\text{clonalg}},\textsc{OneMax}}\right) \geq \left(1 - e^{-\Omega(n)}\right) \cdot \frac{e^{cn}}{n} = e^{\Omega(n)}. \qquad \square$$

It is easy to see from the proof that the optimization time is also $e^{\Omega(n)}$ with overwhelming probability. Moreover, we remark, that already for $\textsc{OneMax}(x_0) = \omega(\log n)$ we have a mutation probability $e^{-\omega(\log n)} = n^{-\omega(1)}$. In this case, the expected number of flipping bits in an iteration is $n/n^{\omega(1)}$, which is also converging to 0 for $n \to \infty$.

Setting $\rho = \ln n$

We have seen, that on one hand for $\rho = \Omega(n)$ the mutation probability becomes exponentially small and thus, yields an exponential expected waiting time for a mutation actually changing a bit. On the other hand for $\rho = O(1)$, the mutation probability becomes quite large, i. e., $1/e \leq p_{\text{CLONALG}}(\textsc{OneMax}(x)) \leq 1$, yielding an exponentially small probability for a successful mutation step once $\textsc{OneMax}(x) \geq cn$ for some constant $c > 1/2$. Again, the algorithm with this parameter setting will not find the global optimum of OneMax within an exponential number of iterations with overwhelming probability.

For these reasons, we now choose a value for ρ, which is in between the two extreme cases, namely $\rho = \ln n$. For $\rho = \ln n$, we have

$$p_{\text{CLONALG}}(\textsc{OneMax}(x)) = e^{-\ln n \cdot \frac{\textsc{OneMax}(x)}{n}} = n^{-\frac{\textsc{OneMax}(x)}{n}},$$

which is $1/n \leq p_{\text{CLONALG}}(\textsc{OneMax}(x)) < 1/2$ if $\textsc{OneMax}(x) > n/\log n$. Hence, in this case we get, in contrast to $\rho = O(1)$ and $\rho = \Omega(n)$, reasonable values for the mutation probability.

In the following we show that also for $\rho = \ln n$ we get indeed a similar result as for $\rho = O(1)$. However, we will later see in Section 4.5 that $\rho = \ln n$ shows a much better performance in practice than the other parameter choices investigated. This is among other reasons due to the fact that the probability not to find the optimum within an exponential number of iterations converges much more slowly to 1 than it does for $\rho = O(1)$.

Theorem 4.8 (Zarges (2008))**.** *Let $\mu = 1$ and $\rho = \ln n$. The optimization time of Algorithm 3.1 using the mutation operator from Algorithm 4.3 and the normalization method from Algorithm 4.4 is*

$$T_{p_{CLONALG},\textsc{OneMax}} \geq e^{d \cdot n^c}$$

with probability at least $1 - e^{-\Omega(n^c)}$ (for constants $0 < c < 1/2$ and $d > 0$ sufficiently small).

Proof. Again, we follow the line of thought of Theorem 4.3. We denote the number of 0-bits in the current search points by z. Moreover, let M_0 be the number of flipping

0-bits and M_1 the number of flipping 1-bits, respectively. By Chernoff bounds (see Lemma B.21), we have $\text{Prob}\left(n/3 \leq \textsc{OneMax}(x_0) \leq 3n/4\right) = 1 - e^{-\Omega(n)}$ for the initial search point x_0. Assume $\textsc{OneMax}(x) = d \cdot n$ for some constant $0 < d < 1$. For $\rho = \ln n$ this yields

$$\begin{aligned} \text{E}\left(M_0\right) &= (1-d) \cdot n^{1-d} \\ \text{E}\left(M_1\right) &= d \cdot n^{1-d}. \end{aligned}$$

We show analogously to Theorem 4.3, that $\text{Prob}\left(\textsc{OneMax}(y) > 3n/4\right) = e^{-\Omega(n)}$ as long as $\textsc{OneMax}(x) \leq 3n/4$. Let δ be some constant with $0 < \delta < 1/6$. We distinguish the following two cases and again repeat the complete calculations due to the different constants involved.

1. $(1/2 + \delta) \cdot n \leq \textsc{OneMax}(x) \leq 3n/4$:

   Similarly to Theorem 4.6 we show a large negative drift in the considered region of the search space. Due to Chernoff bounds, we then have

$$\begin{aligned} \text{Prob}\left(M_0 > (1+\delta) \cdot \text{E}\left(M_0\right)\right) &\leq e^{-\frac{\delta^2 \cdot (1-d) \cdot n^{1-d}}{3}} = e^{-\Omega\left(n^{\frac{1}{2}-\delta}\right)} \text{ and} \\ \text{Prob}\left(M_1 < (1-\delta) \cdot \text{E}\left(M_1\right)\right) &\leq e^{-\frac{\delta^2 \cdot d \cdot n^{1-d}}{2}} = e^{-\Omega\left(n^{\frac{1}{2}-\delta}\right)}. \end{aligned}$$

   For $z = n - \textsc{OneMax}(x) \leq (1/2 - \delta) \cdot n$, we conclude that

$$\begin{aligned} M_0 - M_1 &\leq (1+\delta) \cdot \text{E}\left(M_0\right) - (1-\delta) \cdot \text{E}\left(M_1\right) \\ &= (1+\delta) \cdot (1-d) \cdot n^{1-d} - (1-\delta) \cdot d \cdot n^{1-d} \\ &= n^{1-d} \cdot (1 + \delta - 2d) \\ &\leq n^{1-(1/2+\delta)} \cdot (1 + \delta - 2 \cdot (1/2 + \delta)) \\ &= -\delta \cdot n^{1/2-\delta} < 0 \end{aligned}$$

   holds with probability $1 - e^{-\Omega(n^c)}$. This yields

$$\begin{aligned} &\text{Prob}\left(\textsc{OneMax}(y) > \textsc{OneMax}(x)\right) = \text{Prob}\left(M_0 > M_1\right) \\ &\leq \text{Prob}\left(M_0 > (1+\delta) \cdot E(M_0)\right) \cdot \text{Prob}\left(M_1 < (1-\delta) \cdot E(M_1)\right) = e^{-\Omega(n^c)} \end{aligned}$$

   for the first case.

2. $n/3 \leq \textsc{OneMax}(x) < (1/2 + \delta) \cdot n$:

   In the second case, we show that with overwhelming probability we either stay in the second case or 'jump' to the region of the first case. Note, that the expected progress $\text{E}\left(M_0 - M_1\right)$ is again maximal for $\textsc{OneMax}(x) = n/3$. Let z' be the number of 0-bits after mutating a search point with $(1/2 - \delta) \cdot n < z < 2n/3$ 0-bits and note that $1/3 < d < (1/2 + \delta)$ holds. Due to the above calculations, we get the following lower bound on z'.

$$\begin{aligned} z' &\geq z - n^{1-d} \cdot (1 + \delta - 2d) \\ &> \left(\frac{1}{2} - \delta\right) \cdot n - n^{1-d} \cdot (1 + \delta - 2d) \\ &> \frac{n}{3} - \frac{n^{\frac{2}{3}}}{2} > \frac{n}{4} \qquad \text{for sufficiently large } n \end{aligned}$$

Since $z' > n/4$ implies $\textsc{OneMax}(y) < 3n/4$, we conclude that with probability $1 - 2^{-\Omega\left(n^{\frac{1}{2}-\delta}\right)}$ the function value does not increase beyond $3n/4$.

We have shown that, if $n/3 \leq \textsc{OneMax}(x_0) \leq 3n/4$, $\text{Prob}\left(\textsc{OneMax}(y) > 3n/4\right) = e^{-\Omega\left(n^{\frac{1}{2}-\delta}\right)}$ holds . Together this shows that with probability $1 - e^{-\Omega\left(n^{\frac{1}{2}-\delta}\right)}$ even in $e^{d \cdot n^c}$ iterations the global optimum of $\textsc{OneMax}$ is not found ($0 < c < 1/2$, $d > 0$ sufficiently small, constant). $\square$

Similarly to our analysis in Section 4.2 we can show, that for some regions of the search space we achieve an expected polynomial optimization time.

Theorem 4.9 (Zarges (2008))**.** *Let $\mu = 1$, $\rho = \ln n$ and x_0 be the initial search point. Assume $\textsc{OneMax}(x_0) = n - O(n/\log n)$. The expected optimization time of Algorithm 3.1 using the mutation operator from Algorithm 4.3 and the normalization method from Algorithm 4.4 on* $\textsc{OneMax}$ *is*

$$E(T_{p_{CLONALG}, \textsc{OneMax}, x_0}) = O(n \log n).$$

Proof. Let $k = n - \textsc{OneMax}(x_0)$. For $k = O(n/\log n)$ we get an upper bound for the mutation probability by

$$p_{\text{CLONALG}}(\textsc{OneMax}(x)) = n^{-\frac{n-k}{n}} = \frac{n^{\frac{k}{n}}}{n} = \frac{e^{\frac{\ln(n) \cdot k}{n}}}{n} = O\left(\frac{1}{n}\right).$$

Moreover, we have the trivial lower bound $n^{-(n-k)/n} \geq 1/n$. The probability for a successful one bit mutation can then be bounded below by

$$\binom{k}{1} \cdot \frac{1}{n^{\frac{n-k}{n}}} \cdot \left(1 - \frac{1}{n^{\frac{n-k}{n}}}\right)^{n-1} = \Theta\left(\frac{k}{n}\right) \cdot \left(1 - \Theta\left(\frac{1}{n}\right)\right)^{n-1} = \Theta\left(\frac{k}{n}\right)$$

and the expected waiting time is bounded above by

$$\sum_{k=1}^{O(n/\log n)} \Theta\left(\frac{n}{k}\right) = O(n \log n)$$ $\square$

We have seen that in the very simple framework we are generally only able to show negative results. We claim that this is due to the very restricted normalization method that leads to either too large or too small mutation probabilities. Thus, in the following, we investigate the role a larger collection of search points and introduce a more advanced normalization method.

4.3.2. Normalizing Fitness Values: On the Utility of the Size of the Collection of Search Points

In the foregoing section, we have seen that the mutation operator from CLONALG embedded in a very simple algorithmic framework with only one search point in the collection of search points is not even able to solve the very simple example function OneMax due to a large negative drift. Since typically optimization problems are much harder than OneMax, which can easily be solved by a hill-climbing algorithm, the resulting algorithm does not seem appropriate for the task of optimization. However, we show that this failure is due to the rather artificial restriction of the size of the collection of search points to one and the resulting normalization method embedded in the mutation operator. Thus, in this section, we move forward to larger collections of search points and investigate the performance of the algorithm emerging from this extension.

In the field of evolutionary algorithms, the influence of the size of the collection of search points has been studied by many. In theoretical run time analysis, Jansen and Wegener (2001) and Witt (2008) considered fitness-proportional selection whereas Witt (2006) and Storch (2004) used uniform selection. They showed that larger collection of search points can be advantageous in both selection schemes even if mutation is the only search operator used by the algorithm. We show similar results here.

We first reconsider the normalization method used by the mutation operator. As already discussed previously, in practical applications typically nothing is known about the optimal solution of a problem. Thus, by introducing a larger collection of search points, we can avoid to use this (usually unavailable) information for the purpose of normalization and evaluate the relative fitness of each search point by normalizing by means of the currently best known function value, i. e., the fitness of the best search point in the collection of search points. We define the resulting normalization operator formally for some arbitrary objective function $f\colon \{0,1\}^n \to \mathbb{R}$ in Algorithm 4.5.

Algorithm 4.5 Normalizing by means of the best current known function value f_{best} for $f\colon \{0,1\}^n \to \mathbb{R}^+$.

```
FUNCTION normalize($v$):
1. Let $f_{\text{best}} = \max_{y \in C} f(y)$.
2. Return $v/f_{\text{best}}$.
```

Note, that this normalization method leads to lower mutation probabilities compared to using the optimal value since $\max_{y\in C} f(y)$ is (in case of a maximization problem) always at most as large as the optimal function value. As one reason for the failure of the algorithm analyzed in the previous section are too large mutation probabilities, the normalization method defined in Algorithm 4.5 seems to be a worthwhile concept.

Note, that for $x \in C$ with $f(x) = \max_{y\in C} f(y)$, the corresponding mutation probability equals the standard mutation probability $1/n$ and thus, the behavior of the currently best search point in the collection of search points equals that of the (1+1) EA (Droste et al. 2002). The probability to choose a best search point for reproduction is at least $1/\mu$.

With this observation simple upper bounds on the optimization time can be derived using results for the (1+1) EA. Moreover, some results for the (μ+1) EA (Witt 2006) can be carried over.

Since the normalization method defined above requires function values that are strictly positive, we consider a strictly positive variant of the example function OneMax, called OneMax$^+$, instead. The function can formally be defined as follows. Note, that this modification has no significant effect on the performance of the algorithm and thus, we essentially carry out an analysis on OneMax.

Definition 4.10. *For $n \in \mathbb{N}$ and $x \in \{0,1\}^n$, the function* OneMax$^+$: $\{0,1\}^n \to \mathbb{R}$ *is defined by*

$$\textsc{OneMax}^+(x) = 1 + \textsc{OneMax}(x) = 1 + \sum_{i=0}^{n-1} x[i].$$

In the previous section, we found out that the choice of the decay parameter ρ is essential for the performance of the algorithm and identified $\rho = \ln n$ as an appropriate choice that leads at least to reasonable mutation probabilities, i. e., $1/n \leq p_{\text{CLONALG}} < 1/2$ if $\textsc{OneMax}(x) > n/\log n$. Note, that an optimal choice for ρ is problem-dependent and that it is not known whether $\ln n$ is the best value for the considered objective function. Nevertheless, due to our former results for OneMax we set $\rho = \ln n$ throughout this section. This leads to the following mutation probability for a strictly positive pseudo-Boolean objective function $f\colon \{0,1\}^n \to \mathbb{R}^+$:

$$p_{\text{CLONALG}}(f(x)) = e^{-\ln n \cdot \frac{f(x)}{\max_{y \in P} f(y)}} = n^{-\frac{f(x)}{\max_{y \in P} f(y)}}. \tag{4.6}$$

In the following, we consider upper and lower bounds of Algorithm 3.1 with $\mu \geq 2$ and the mutation operator described above. Along the way we derive some general characteristics of the considered mutation operator as well as properties of the collection of search points, which hold not only for OneMax but for a larger class of pseudo-Boolean functions.

An Upper Bound for OneMax

We first derive an upper bound for the optimization time of Algorithm 3.1 using the mutation operator from Algorithm 4.3 with $\rho = \ln n$ and the normalization method defined in Algorithm 4.5 on OneMax$^+$. This bound easily carries over from the analysis of the (μ+1) EA executed by Witt (2006) and thus, we are able to show an asymptotically tight polynomial bound on the expected optimization time for the considered algorithm. This shows that larger collections of search points can be essential for inversely fitness-proportional mutation probabilities since (as seen in the previous section) the respective algorithm with $\mu = 1$ is inefficient when optimizing OneMax$^+$.

We remark, that the essential part is the normalization method incorporated in the algorithm which only makes sense when the collection of search points consists of at least two search points. However, if $\mu = 1$ is chosen in this setting, the resulting algorithm equals the (1+1) EA with mutation probability $1/n$ (and not the respective algorithm from the previous section) since always the currently best search point in the collection of search points, namely the only one, is selected and thus, the mutation probability in this case is $1/n$ throughout the whole optimization process. We see that for this reason with $\mu = 1$ the resulting algorithm is not an immune algorithm in our sense as the mutation probability does not depend on the function value any more. As the focus of our analysis lies on immune algorithms and especially on the dynamics of the considered mutation operator, we exclude the case $\mu = 1$ from our considerations for the rest of this section.

Theorem 4.11 (Zarges (2009))**.** *Let $\rho = \ln n$, $\mu = poly(n)$ and $\mu \geq 2$. The expected optimization time of Algorithm 3.1 using the mutation operator from Algorithm 4.3 and the normalization method from Algorithm 4.5 on* OneMax$^+$ *is*

$$E\left(T_{pCLONALG,\text{OneMax}^+}\right) \leq 5e\mu n + en\ln(en) = O(\mu n + n\log n).$$

Proof. Note, that in the proof by Witt (2006; Theorem 2) only the currently best search point in the collection of search points is considered, which implies that the relevant mutation probability for our algorithm equals $1/n$. Thus, the proofs follows exactly the argumentation carried out by Witt (2006) for the (μ+1) EA. We remark, that the bound in Witt (2006) had an extra additive term of μ which reflects the costs of initialization. However, we only count iterations here (in contrast to function evaluations) and thus, these costs are 0 in our cost model.

Let L be the number of 1-bits in the currently best known search point x, i. e., $L = |x|_1$. To increase L and thus, increase the currently best known function value, it suffices to select one of the best search points from the current collection of search points and flip exactly one of the existing 0-bits. If there exist i best search points, this probability is

$$\frac{i}{\mu} \cdot \frac{n-L}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{i \cdot (n-L)}{e\mu n}.$$

Assume pessimistically that L does not increase until the collection of search points contains at least $\min\{n/(n-L), \mu\}$ best search points. The number of best search points can increase if Algorithm 3.1 produces replicas of best search points. The probability for this event is

$$\frac{i}{\mu} \cdot \left(1 - \frac{1}{n}\right)^{n} \geq \frac{i}{2e\mu}.$$

Thus, the expected waiting time for having at least $\min\{n/(n-L), \mu\}$ best search points is at most

$$\sum_{i=1}^{\lceil n/(n-L)\rceil - 1} \frac{2e\mu}{i} \leq 2e\mu \ln\left(\frac{en}{n-L}\right)$$

if L does not increase before. Summing up these expecting waiting times for all values of L yields a total expected waiting time of at most

$$2e\mu \sum_{L=0}^{n-1} \ln\left(\frac{en}{n-L}\right) = 2e\mu \ln\left(\frac{e^n n^n}{n!}\right) \leq 2e\mu \ln(e^{2n}) = 4e\mu n.$$

Moreover, if there are at least $\min\{n/(n-L), \mu\}$ best search points, the expected time for increasing L is at most $e\mu n/\min\{n, \mu \cdot (n-L)\}$ and thus, the expected waiting time for all L-increases is at most

$$\sum_{L=0}^{n-1} \left(\frac{e\mu n}{\mu(n-L)} + \frac{e\mu n}{n} \right) \leq en\ln(en) + e\mu n.$$

Altogether, the total expected optimization time is at most $en\ln(en) + 5e\mu n$. □

Note, that for $\mu = \Theta(1)$ the upper bound on the optimization time of the considered algorithms is asymptotically the same as for the (1+1) EA (Droste et al. 2002). Thus, already for $\mu = 2$ the performance of the considered mutation operator on simple example functions is drastically improved compared to the parameterization chosen in the previous section.

Deriving a Matching Lower Bound

We now derive an asymptotically equivalent lower bound. Again, we follow the argumentation carried out by Witt (2006) for the (μ+1) EA. However, for the lower bound, we require additional knowledge of the properties of the considered mutation operator. Since the mutation probability directly depends on the structure of the collection of search points, we first investigate general properties of this structure. To be more precise, we derive an upper bound on the size of the span of the function values within the collection of search points. By means of this result, we show that the expected number of flipping bits in an iteration is bounded by constants. Afterwards, we can adapt the arguments by Witt (2006).

The results of this section are taken from Zarges (2009) and hold for a class of pseudo-Boolean functions where the function values only depend on the number of 1-bits in the search points. Moreover, the functions in this class have somehow *smooth* fitness landscapes, i. e., neighboring points in the search space have similar function values. These functions were already investigated by Jansen and Wegener (2007). Note, that we again only consider strictly positive functions due to the normalization method embedded in the mutation operator. Clearly, OneMax$^+$ belongs to this class. We define both properties formally.

Definition 4.12. *A pseudo-Boolean function $f\colon \{0,1\}^n \to \mathbb{R}^+$ is called* smooth integer *if $f(x) \in \mathbb{Z}^+$ for all $x \in \{0,1\}^n$ and if $|f(x) - f(y)| \leq c$ ($c > 0$ constant) for all $x, y \in \{0,1\}^n$ with $H(x,y) \leq 1$.*

Definition 4.13. *The class U_n contains all pseudo-Boolean functions $f\colon \{0,1\}^n \to \mathbb{R}^+$ of unitation, where $f(x)$ depends only on the number of 1-bits in x. The functions in U_n are also known as symmetrical functions since the function value does not change by permuting the input bits.*

The mutation probability for a search point x depends on the relation of the function value of x and the currently best known function value. We observe, that the initial collection of search points of the algorithm is, with high probability, situated within *central* regions of the search space, i. e., search points with $n/2 \pm O(\sqrt{n \log n})$ 1-bits. Thus, we additionally characterize functions by a lower bound on the function values in these central regions in order to estimate the relation between the best and worst function value within the collection of search points.

Definition 4.14. *A pseudo-Boolean function $f\colon \{0,1\}^n \to \mathbb{R}^+$ is denoted as $\kappa(n)$-*qualified *if for all $x \in \{0,1\}^n$ with $n/2 - O\big(\sqrt{n \cdot \log n}\big) \leq \textsc{OneMax}(x) \leq n/2 + O\big(\sqrt{n \cdot \log n}\big)$ we have $f(x) = \Omega(\kappa(n))$.*

Note, that $\textsc{OneMax}^+$ is n-qualified. We start with a general result on the considered mutation operator and show that the expected number of flipping bits in an iteration is bounded by constants if the difference between the function values of the best and worst search point in the current collection of search points is at most $O(n/\beta(n))$ and the best search point has function value at least $\Omega(n \log n/\beta(n))$ for an arbitrary $\beta(n)$. Afterwards, we show that the prerequisites of the following lemma hold for the considered algorithm with high probability.

Lemma 4.15 (Zarges (2009))**.** *Consider the mutation probability defined in eq. (4.6). Let $f\colon \{0,1\}^n \to \mathbb{R}^+$ and C a collection of search points with*

$$v_{\max} = \max_{x \in C} f(x) = \Omega\left(\frac{n \cdot \log n}{\beta(n)}\right) \text{ and}$$
$$v_{\min} = \min_{x \in C} f(x) = v_{\max} - O\left(\frac{n}{\beta(n)}\right)$$

for an arbitrary $\beta(n)$. Then, for any search point from C the expected number of flipping bits is $\Theta(1)$.

The probability to flip $\omega(\gamma(n))$ bits is bounded above by $e^{-\omega(\gamma(n))}$. If $\gamma(n) = \Omega(\log n)$, this even holds for a sequence of $T = poly(n)$ independent mutations.

Proof. Let $x \in C$ with $f(x) = v_{\min}$. If x is selected for reproduction, we get

$$p_{\text{CLONALG}}(f(x)) = n^{-\frac{f(x)}{\max_{y \in C} f(y)}} = n^{-\frac{v_{\min}}{v_{\max}}} = n^{-\frac{v_{\max} - O\left(\frac{n}{\beta(n)}\right)}{v_{\max}}} = n^{-\left(1 - \frac{O\left(\frac{n}{\beta(n)}\right)}{v_{\max}}\right)}.$$

Let M be the number of flipping bits. As $v_{max} = \Omega(n \log n/\beta(n))$, the expected number of flipping bits can be bounded above by

$$\mathrm{E}\,(M) = \frac{n}{n^{1 - \frac{O\left(\frac{n}{\beta(n)}\right)}{v_{\max}}}} = n^{\frac{O\left(\frac{n}{\beta(n)}\right)}{v_{\max}}} = n^{\frac{O\left(\frac{n}{\beta(n)}\right)}{\Omega\left(\frac{n \cdot \log n}{\beta(n)}\right)}} = n^{O\left(\frac{1}{\log n}\right)} = 2^{O(1)} = O(1).$$

Moreover, $\mathrm{E}\,(M) \geq 1$ holds as $p_{\text{CLONALG}}(f(x)) \geq 1/n$. Let $\mathrm{E}\,(M) = c$. We bound the probability to flip at least $\omega(\gamma(n))$ bits from above using Chernoff bounds as in the proof of Theorem 4.4 by $2^{-\omega(\gamma(n))}$. Assume $\gamma(n) = \Omega(\log n)$. Then, the probability that this does not hold for one mutation in a sequence of $T = \text{poly}\,(n)$ mutations is bounded above $T/e^{\omega(\gamma(n))} = e^{-\omega(\gamma(n))}$.

Since p_{CLONALG} is monotonically decreasing, the above argumentation holds for all $y \in C$ with $f(y) \geq f(x) = v_{\min}$. □

In order to facilitate notation, we characterize iterations of the considered algorithm according to the progress made, i. e., the improvement of the currently best known function value. Recall, that we denote the collection of search points in iteration t by C_t.

Definition 4.16. *Iteration t of Algorithm 3.1 is called* improving *if* $\max_{x \in C_t} f(x) > \max_{x \in C_{t-1}} f(x)$ *holds and* non-improving *otherwise.*

In a first step we show that the upper bound on the difference in function values given in Lemma 4.15 holds with high probability within phases of the considered algorithm that contain at most μ improving iterations.

Lemma 4.17 (Zarges (2009))**.** *Let C_t be the collection of search points of Algorithm 3.1 using the mutation operator from Algorithm 4.3 and the normalization method from Algorithm 4.5 at time t. Let $\mu = O(n/\beta(n))$ and $\mu \geq 2$ with $\beta(n) = O\big(n/\log^2 n\big)$. Furthermore, let $f\colon \{0,1\}^n \to \mathbb{R}^+$ be smooth integer and $(n \log n/\beta(n))$-qualified. Consider a phase of this algorithm of length $T = \text{poly}\,(n)$ that contains at most μ improving iterations.*

If $|f(x) - f(y)| = O(n/\beta(n))$ for all $x, y \in C_t$, then, with probability $1 - e^{-\omega(\log n)}$, for all $x', y' \in C_{t+i}$, $0 < i \leq T$, $|f(x') - f(y')| = O(n/\beta(n))$ holds.

Proof. Note, that only improving iterations can increase the difference in function values of the best and worst search point in a collection of search points. In the following, we therefore only consider improving iterations and ignore the effects of non-improving iterations since they can only decrease the considered difference.

Let t' be the point in time where the first improving iteration takes place. As $|f(x) - f(y)| = O(n/\beta(n))$ for all $x, y \in C_t$ and all iterations between t and t' are non-improving, we have $|f(x) - f(y)| = O(n/\beta(n))$ for all $x, y \in C_{t'-1}$. Since f is $(n \log n/\beta(n))$-qualified and due to the elitist selection of the algorithm, additionally $f(x) = \Omega(n \log n/\beta(n))$ holds with probability $1 - 2^{-\Omega(n)}$. Thus, in this situation, we can apply Lemma 4.15.

Assume $O(\log n)$ bits are flipped in iteration t'. Due to Lemma 4.15 the probability for this event is bounded below by $1 - e^{-\omega(\log n)}$. As the considered objective function is smooth integer, the number of flipping bits is asymptotically an upper bound on the progress obtained and hence, the improvement is $O(\log n)$ with probability $1 - e^{-\omega(\log n)}$. We obtain, with probability $1 - e^{-\omega(\log n)}$,

$$|f(x) - f(y)| = O\left(\frac{n}{\beta(n)}\right) + O(\log n) = O\left(\frac{n}{\beta(n)}\right)$$

for all $x, y \in C_{t'}$, where the last equality holds due to the prerequisite $\beta(n) = O\big(n/\log^2 n\big)$. We now prove our claim by the following case distinction.

For the first case assume $\mu = O(n/(\beta(n)\log n))$ and let T be the point in time where the μ-th improving iteration takes place. The argumentation from above can be carried over iteratively to the subsequent improving iterations of the considered phase. Altogether, with probability $1 - e^{-\omega(\log n)}$, the difference increases at most by $\mu \cdot O(\log n)$ within μ improving iterations. We then have, with probability $1 - e^{-\omega(\log n)}$,

$$|f(x) - f(y)| = O\left(\frac{n}{\beta(n)}\right) + \mu \cdot O(\log n)$$
$$= O\left(\frac{n}{\beta(n)}\right) + O\left(\frac{n}{\beta(n)\cdot\log n}\right)\cdot O(\log n) = O\left(\frac{n}{\beta(n)}\right)$$

for all $x, y \in C_T$, where the second equality holds due to the assumption on μ in this case.

For the second case, let $\mu = \Omega(n/(\beta(n)\log n))$. As stated above, we can apply Lemma 4.15 and thus, the expected number of flipping bits in one iteration is $\Theta(1)$. Since each bit is flipped independently with the same mutation probability p and we have n bits we conclude that $p = \Theta(1/n)$ holds. We consider μ iterations. The expected number of flipping bits is $\Theta(\mu \cdot n \cdot (1/n)) = \Theta(\mu)$. Due to Chernoff bounds, the probability to flip $\omega(\mu)$ bits is then bounded above by $e^{-\omega(\mu)}$. Hence, using $\beta(n) = O(n/\log^2 n)$, with probability

$$1 - e^{-\omega(\mu)} = 1 - e^{-\omega\left(\frac{n}{\beta(n)\cdot\log n}\right)} = 1 - e^{-\omega(\log n)},$$

the difference increases at most by $O(\mu)$ within μ improving iterations. With $\mu = O(n/\beta(n))$ the lemma follows. □

We see that the argumentation of the proof of Lemma 4.17 cannot be carried forward to a larger number of improving iterations: For λ improving iterations the resulting upper bound on the difference in function values is $O(n/\beta(n)) + \lambda \cdot O(\log n)$ or $O(n/\beta(n)) + O(\lambda)$ respectively, and we need this to be $O(n/\beta(n))$. Thus, to obtain a general result for the considered algorithm with a polynomial number of iterations, we need to analyze the development of the collection of search points more carefully. Due to notational convenience we introduce the following notion of *subsequent* search points within a *sorted* collection of search points.

Definition 4.18. *Let $C_t = \{x_1, x_2, \ldots, x_\mu\}$ be the collection of search points of the Algorithm 3.1 at time t sorted according to the function values, i. e., $f(x_i) \le f(x_j)$ for $i < j$. Then, x_i and x_{i+1} are denoted as* subsequent *search points. Moreover, we call C_t* sorted.

The idea for the proof of the following main theorem on the properties of the collection of search points is as follows. We first show that our claim on the difference in function values holds after initialization. Due to Lemma 4.17 this property is not destroyed within μ improving iterations. We therefore consider phases of Algorithm 3.1 with exactly μ improving iterations and show that after each of these phases, the collection of search points is repeatedly situated within a narrow band, i. e., we can derive an upper bound on the difference of the function values of two subsequent search points. Thus, we are able to apply inductive arguments. Note, that the last such phase might contain less than μ improving iterations. As this is not crucial for the proof, we assume to only have phases with exactly μ improving iterations in the following.

Theorem 4.19 (Zarges (2009)). *Let C_t be a collection of search points of Algorithm 3.1 using the mutation operator from Algorithm 4.3 and the normalization method from Algorithm 4.5 at time t, $0 \leq t \leq T$, $T = \text{poly}(n)$, $\mu = O(n/\beta(n))$, $\mu \geq 2$, with $\beta(n) = O\left(\sqrt{n/\log n}\right)$ and $x, y \in C_t$.*

Then, for a $(n \log n/\beta(n))$-qualified smooth integer function of unitation $f\colon \{0,1\}^n \to \mathbb{R}^+$ we have $|f(x) - f(y)| = O(n/\beta(n))$ with probability $1 - e^{-\omega(\log n)}$.

Proof. After initialization, we have $n/2 - O(n/\beta(n)) \leq |x|_1 \leq n/2 + O(n/\beta(n))$ for a single search point x with probability $1 - e^{-\omega\left(n/\beta(n)^2\right)}$ due to Chernoff bounds (Lemma B.21). The probability that this does not hold for one of the μ search points is bounded above by $\mu \cdot e^{-\omega\left(n/\beta(n)^2\right)}$ using a simple union bound (Lemma B.10). For $\mu = \text{poly}(n)$, this probability can be bounded above by $e^{-\omega(\log n)}$ since due to the prerequisite $\beta(n) = O\left(\sqrt{n/\log n}\right)$, $1 - e^{-\omega\left(n/\beta(n)^2\right)} = 1 - e^{-\omega(\log n)}$ holds.

As f is a function of unitation the number of different function values for these initial search points is bounded above by $2 \cdot O(n/\beta(n)) = O(n/\beta(n))$. Let $x = 1^a 0^{n-a}$ and $y = 1^b 0^{n-b}$ be two search points with maximal difference in the number of 1-bits. As there exists a path of direct Hamming neighbors between x and y of length $O(n/\beta(n))$ and f is smooth integer, the difference in function values between x and y is bounded above by $O(n/\beta(n))$. Since f is a function of unitation, this holds for all search points after initialization with probability $1 - e^{-\omega(\log n)}$ and thus, the claim holds after initialization.

Assume t' to be the point in time where the μ-th improving iteration takes place. As $\beta(n) = O\left(\sqrt{n/\log n}\right) = O\left(n/\log^2 n\right)$ we can apply Lemma 4.17. Thus, we have $|f(x) - f(y)| = O(n/\beta(n))$ for all $x, y \in C_t$, $0 \leq t \leq t'$, with probability $1 - e^{-\omega(\log n)}$.

We further investigate such an iteration t, i. e., $0 \leq t \leq t'$. Let $v_{\min} = \min_{x \in C_{t-1}} f(x)$ be the minimal and $v_{\max} = \max_{x \in C_{t-1}} f(x)$ be the maximal function value before iteration t. If t is improving, definitely a new search point enters the collection of search points, which potentially increases the difference of the function values of the best and worst search point as $v_{\max}$ increases. If t is non-improving, also a new search point y might enter the collection of search points, but due to the elitist selection of the algorithm and as t is non-improving, in this case $v_{\min} \leq f(y) \leq v_{\max}$ holds. Thus, the difference in function values never increases but decreases iff $v_{\min}$ increases. Altogether, the difference in function values is maximal if we only consider improving iterations and thus, the collection of search points $C_{t'}$ contains, in the worst case, exactly the μ search points that were created during the μ improving iterations of the considered phase. Hence, $\max_{x \in C_0} f(x) < \min_{x \in C_{t'}} f(x)$.

Analogously to Lemma 4.17 we prove the theorem by the following case distinction.

For the first case assume $\mu = O(n/(\beta(n)\log n))$. Due to Lemma 4.15 the progress obtained in a single improving iteration in a sequence of $T = \text{poly}\,(n)$ iterations is $O(\log n)$ with probability $1 - e^{-\omega(\log n)}$. This implies that the difference of the function values of two subsequent search points of the collection of search points is bounded above by $O(\log n)$, i. e., $|f(x_{i+1}) - f(x_i)| \le c \cdot \log(n)$, $c > 0$ constant, with probability $1 - e^{-\omega(\log n)}$. Note, that c is the same for all pairs of subsequent search points. With $\mu = O(n/(\beta(n)\log n))$, say $\mu = d \cdot n/(\beta(n)\log n)$ for some constant $d > 0$, the difference of the function values of x_1 and x_μ can be bounded above by

$$\begin{aligned}\sum_{i=1}^{\mu-1} |f(x_{i+1}) - f(x_i)| \le \sum_{i=1}^{\mu-1} c \cdot \log n = \mu \cdot c \cdot \log n\\ = d \cdot \frac{n}{\beta(n) \cdot \log n} \cdot c \cdot \log n = c \cdot d \cdot \frac{n}{\beta(n)} = O\left(\frac{n}{\beta(n)}\right)\end{aligned}$$

For $\mu = \Omega(n/(\beta(n)\log n))$, we get $|f(x_1) - f(x_\mu)| = \delta \cdot \mu = \delta' \cdot \frac{n}{\beta(n)} = O(n/\beta(n))$ with probability at least $1 - e^{-\omega(\log n)}$ by the same argumentation as in the proof of Lemma 4.17 ($\delta, \delta' > 0$ constant).

For the rest of the algorithm, we now consider phases that contain exactly μ improving iterations. Due to Lemma 4.17 the stated property survives within each of these phases.

Let t_1^i be the beginning and t_2^i the end of phase i. If $|f(x) - f(y)| = O(n/\beta(n))$ for all $x, y \in C_{t_1^i}$, we have $|f(x) - f(y)| = O(n/\beta(n))$ for all $x, y \in C_t$, $t_1^i \le t \le t_2^i$, with probability $1 - e^{-\omega(\log n)}$ due to Lemma 4.17. Moreover, the collection of search points at time t_2^i again contains, in the worst case, exactly the μ search points that were created during the μ improving iterations of the phase. Therefore, after each of the considered phases the difference in function values between x_1 and x_μ can be bounded above in exactly the same way, i. e., by $cdn/\beta(n)$ or $\delta' n/\beta(n)$ respectively for constants $c, d, \delta' > 0$ from the calculations performed above. This concludes the proof. $\square$

Using the results from Theorem 4.19 and Lemma 4.15 we can now easily derive an upper bound on the expected number of flipping bits in an iteration of the considered algorithm as long as the total number of iterations is polynomial in n. More precisely, we show that the expected number of flipping bits in an iteration is constant. Afterwards, we use this result to prove the lower bound on the optimization time.

Corollary 4.20 (Zarges (2009)). *Let $f\colon \{0,1\}^n \to \mathbb{R}^+$ be a $(n\log n/\beta(n))$-qualified smooth integer function of unitation, $\beta(n) = O\left(\sqrt{n/\log n}\right)$, and C_t a collection of search points evolved by Algorithm 3.1 using the mutation operator from Algorithm 4.3 and the normalization method from Algorithm 4.5 at time t with $\mu = O(n/\beta(n))$ and $\mu \ge 2$.*

Then, for an arbitrary search point from C_t the expected number of flipping bits in an iteration is $\Theta(1)$ as long as $t = poly\,(n)$.

Proof. Due to Theorem 4.19 we have $|f(x) - f(y)| = O(n/\beta(n))$ for arbitrary $x, y \in C_t$ with probability at least $1 - e^{-\omega(\log n)}$. As f is $(n\log n/\beta(n))$-qualified and due to the elitist selection of the considered algorithm, additionally $f(x) = \Omega(n \log n/\beta(n))$ for all $x \in C_t$ holds with probability $1 - 2^{-\Omega(n)}$. Thus, in this situation, the expected number of flipping bits is $c \geq 1$ for some $c = \Theta(1)$ according to Lemma 4.15.

For $|f(x) - f(y)| = \omega(n/\beta(n))$, we can bound the number of flipping bits by n. Thus, this case adds $n/e^{\omega(\log n)} = o(1)$ to the total expected value and the corollary follows. □

We remark, that the results presented above also hold for the $(\mu+1)$ EA (Witt 2006). Moreover, both Lemma 4.17 and Theorem 4.19 only consider improving iterations. Thus, we only consider iterations that increase the difference in function values of the best and worst search point and ignore iterations that might lead to reductions of the difference in function values. In spite of our worst case argumentation the results obtained are surprisingly good, but may leave room for improvements.

However, we are now ready to prove the announced lower bound on the optimization time. The following proof uses the technique of analyzing randomized family trees introduced by Witt (2006). For each search point x_0 in the initial collection of search points a family tree contains this search point x_0 as its root and all its descendants as nodes. Some node x in a family tree gets a child node x' if x' is generated via mutation of x. Removal of search points from the collection of search points is not modeled in family trees. Nodes in a family tree that correspond to members of the current collection of search points are called *alive*. Other nodes are called *dead*. Clearly, only alive nodes can get new descendants. The *depth* of a node in a family tree corresponds to the number of mutations between the node and its oldest ancestor at the root. Since mutations tend to be small, probabilistic relations between the depth of nodes and their Hamming distance to the root can be established. If the depth of a node in a family tree is not too small it can be expected that it evolved far. Thus, lower bounds on the depth of a family tree can be translated into upper bounds on the optimization time. Analogously, upper bounds on the depth of a family tree translate into lower bounds on the optimization time. This makes family trees an immensely useful proof method for collection of search points-based search heuristics. We continue with a more formal definition.

A family tree $T_t(x_0)$ contains the direct and indirect descendants of a search point x_0 created by time $t \geq 0$ via mutation. Thereby, nodes of the tree identify the search points generated and an edge (x, y) denotes that y was created by mutating x. At any time step there is a family tree $T_t(x_0)$ for each x_0 from the initial collection of search points. The depth of a family tree is defined as the maximal depth of its nodes. The depth of a node is defined as the number of nodes encountered on the unique path from the root to this node not including the node itself. We remark, that we use this technique again with a slight modification in Section 9.2 when considering different aging operators. There, an example for a family tree can be found (Figure 9.3, page 172).

Analogously to Witt (2006), we start with a general lower bound for $(n \log n/\beta(n))$-qualified smooth integer functions of unitation whose set of global optima have size $2^{o(n)}$ or where all global optima have a linear Hamming distance to search points with $n/2$ 1-bits. Finally, we derive a lower bound on the optimization on our concrete objective function OneMax$^+$ (see Definition 4.10).

Theorem 4.21 (Zarges (2009))**.** *Let $\mu = O(n/\beta(n))$ with $\beta(n) = O\Big(\sqrt{n/\log n}\Big)$, $\mu \geq 2$, and $f\colon \{0,1\}^n \to \mathbb{R}^+$ $(n\log n/\beta(n))$-qualified smooth integer of unitation. Moreover, let $OPT = \{x \in \{0,1\}^n \mid f(x) = \max\{f(x') \mid x' \in \{0,1\}^n\}\}$ denote the set of all global optima with $o_{\min} = \min\{|x|_1 \mid x \in OPT\}$ and $o_{\max} = \max\{|x|_1 \mid x \in OPT\}$. If at least one of the three conditions*

$$|OPT| = 2^{o(n)}, \quad o_{\min} = \frac{n}{2} + \Omega(n), \quad o_{\max} = \frac{n}{2} - \Omega(n)$$

holds, the expected optimization time of Algorithm 3.1 using the mutation operator from Algorithm 4.3 and the normalization method from Algorithm 4.5 is at least $\Omega(\mu n)$, and the success probability in $c\mu n$ steps is $2^{-\Omega(n)}$ if the constant $c > 0$ is chosen sufficiently small. If $|OPT| = 1$, the expected optimization time on f is even $\Omega(\mu n + n\log n)$.

Proof. Since all requirements of Corollary 4.20 are fulfilled, we can conclude that the expected number of flipping bits in an iteration of the considered algorithm is constant. Thus, the proof of Theorem 4 by Witt (2006) can be carried over by simply adjusting the calculations performed.

We start with the lower bound $\Omega(n\log n)$ in the case $|\mathrm{OPT}| = 1$. This bound needs only to be shown for $\mu \leq c'\log n$ for an arbitrary constant $c' > 0$. Let $c' = 1/2$. Due to Corollary 4.20 the expected number of flipping bits in an iteration is $d \geq 1$ for some $d = \Theta(1)$ and thus, the mutation probability for a single bit equals d/n. The probability that bit $i \in \{1,\dots,n\}$ is different from the unique optimal assignment in all initial search points is at least $n^{-1/2}$. Due to Chernoff bounds (Lemma B.21), at least $\sqrt{n}/2$ bits are wrong in all initial search points with probability $1 - 2^{-\Omega(\sqrt{n})}$. Assuming that there are $\sqrt{n}/2$ such bits, the probability that at least one of these bits never flips within $t = \lfloor (n/d - 1)\cdot(\ln n)/2\ \rfloor$ steps is bounded below by

$$1 - \left(1 - \left(1 - \frac{d}{n}\right)^t\right)^{\sqrt{n}/2} \geq 1 - \left(1 - \frac{1}{\sqrt{n}}\right)^{\sqrt{n}/2} \geq 1 - e^{-1/2}$$

which implies that t steps are required with probability at least $1 - e^{-1/2} - 2^{-\Omega(\sqrt{n})}$ $= \Omega(1)$. Hence, the lower bound $\Omega(n\log n)$ on the expected optimization time follows.

We now prove the lower bound $\Omega(\mu n)$ under the three conditions by considering phases of length $s := \lfloor c\mu n \rfloor$ ($c > 0$ constant) and show that the considered algorithm requires at least s steps with probability $1 - 2^{-\Omega(n)}$ if c is small enough. As already noted before, we use the technique of analyzing randomized family trees (Witt 2006).

Let x_0 be an arbitrary initial search point and let $T_t(x_0)$ denote its random family tree at time t. Witt (2006) showed that the probability for $T_s(x_0)$ reaching depth greater than $\lfloor 3cn \rfloor$ is $2^{-\Omega(n)}$. We now show that also with probability $2^{-\Omega(n)}$, there is a node in $T_s(x_0)$ at depth at most $3cn$, which is labeled with a search point whose Hamming distance to x_0 is bounded above by $8cdn$ ($d \geq 1$, constant). Afterwards, we show that the Hamming distance of the root search point to all optimal search points is larger with overwhelming probability.

Consider a sequence of $\lfloor 3cn \rfloor$ points where each point is the result of a mutation of its predecessor by means of the mutation operator from Algorithm 4.3 using the normalization method from Algorithm 4.5. Due to Corollary 4.20 the expected number of flipping bits for each point in this sequence is $d \geq 1$ for some $d = \Theta(1)$. Thus, the expected difference of any two points in the sequence is at most $3cdn$. Due to Chernoff bounds (with respect to the upper bound $4cdn$), the probability to have a Hamming distance of at least $8cdn$ is bounded above by $e^{-4cdn/3}$.

We call a path *bad* if it starts at the root of $T_s(x_0)$, has length $\ell \leq 3cn$ and contains a label with Hamming distance at least $8cdn$ with respect to x_0. The probability to create a specific path of length ℓ with labels $x_0, x_1, \ldots, x_\ell$ is

$$\prod_{i=0}^{\ell-1} \frac{\text{Prob}\left(\text{mut}(x_i) = x_{i+1}\right)}{\mu} = \left(\frac{1}{\mu}\right)^{\ell} \cdot \prod_{i=0}^{\ell-1} \text{Prob}\left(\text{mut}(x_i) = x_{i+1}\right) \leq \left(\frac{1}{\mu}\right)^{\ell}$$

where $\text{Prob}\left(\text{mut}(x) = y\right)$ denotes the probability of creating $y \in \{0,1\}^n$ from $x \in \{0,1\}^n$ by means of mutation. Thus, the probability to create a specific bad path is bounded above by $(1/\mu)^{\ell} \cdot e^{-4cdn/3}$. Since the number of paths of length at most ℓ is bounded above by $\binom{s}{\ell}$, the probability that $T_s(x_0)$ contains a bad path can be estimated by

$$\sum_{\ell=1}^{3cn} \binom{s}{\ell} \cdot \left(\frac{1}{\mu}\right)^{\ell} \cdot e^{-4cdn/3} \leq 3cn \cdot \max_{\ell=\{1,\ldots,3cn\}} \left\{ \left(\frac{ce\mu n}{\ell}\right)^{\ell} \cdot \left(\frac{1}{\mu}\right)^{\ell} \cdot e^{-4cdn/3} \right\}$$

$$= 3cn \cdot e^{-4cdn/3} \cdot \max_{\ell=\{1,\ldots,3cn\}} \left(\frac{cen}{\ell}\right)^{\ell} \leq 3cn \cdot e^{-4cdn/3} \cdot e^{cn} \overset{d\geq 1}{\leq} 3cn \cdot e^{-cn} \overset{c\geq 0}{=} 2^{-\Omega(n)}$$

We finally show that, under the three conditions, the Hamming distance of x_0 to all optimal search points is, with overwhelming probability, larger $8cdn$ if c is small enough. For any $y \in \text{OPT}$, the expected Hamming distance of x_0 to y is $n/2$. Due to Chernoff bounds, the Hamming distance is at least $n/3$ with probability $1 - 2^{-\Omega(n)}$. If $|\text{OPT}| = 2^{o(n)}$, the Hamming distance of x_0 to all $y \in \text{OPT}$ is also at least $n/3$ with probability $1 - 2^{-\Omega(n)}$. For $o_{\min} = n/2 + \Omega(n)$, Chernoff bounds with respect to the expectation $\text{E}\left(|x_0|\right) = n/2$ yield a Hamming distance of at least $\Omega(n)$ with probability $1 - 2^{-\Omega(n)}$. The case $o_{\max} = n/2 - \Omega(n)$ is symmetrical. Thus, in all three cases, choosing c small enough results in $8cdn$ being smaller than the lower bound $\Omega(n)$.

Altogether, for a fixed initial search point x_0, $T_s(x_0)$ does with probability $1 - 2^{-\Omega(n)}$ not contain nodes labeled with optimal search points. As $\mu = \text{poly}\,(n)$, this also holds for all initial search points together. □

Clearly, OneMax$^+$ meets the requirements given in Theorem 4.21 since it is a smooth integer function of unitation, n-qualified and has a single optimum. In addition, the optimum has even linear Hamming distance to search points with $n/2$ 1-bits. Thus, with $\beta(n) = \log n$ the following corollary follows.

Corollary 4.22 (Zarges (2009)). *Let $\mu = O(n/\log n)$ and $\mu \geq 2$. The expected optimization time of Algorithm 3.1 using the mutation operator from Algorithm 4.3 and the* *normalization method from Algorithm 4.5 on* OneMax$^+$ *is*

$$E\left(T_{p_{CLONALG},\textsc{OneMax}^+}\right) = \Theta(\mu n + n \log n).$$

Altogether, we have proven an asymptotically tight bound on the optimization time of Algorithm 3.1 using the mutation operator from Algorithm 4.3 and the normalization method from Algorithm 4.5 on OneMax$^+$ if $\mu = O(n/\log n)$. Moreover, this bound is asymptotically the same as the corresponding bound for the $(\mu{+}1)$ EA.

4.4. The Mutation Operator from opt-aiNet

The mutation operator introduced in opt-aiNet (de Castro and Timmis 2002b) is in some sense quite similar to that in CLONALG. Actually it is based on the immune network theory, but it also incorporates ideas from the clonal selection principle. It also uses an inverse exponential function to establish a relationship between the mutation probability and the normalized function value of the search point to be mutated. Again a parameter ρ controls the appearance of the mutation probability. However, the parameter ρ is not incorporated in the exponent of the exponential function but rather used to scale the result of this function. This leads to a completely different impact of the parameter and the optimization process in general. We define the mutation operator formally in Algorithm 4.6.

Algorithm 4.6 Opt-aiNet mutation probability p_{aiNet} (de Castro and Timmis 2002b).

```
FUNCTION mutate(x):
1. Let $v :=$ normalize$(f(x)) \in [0,1]$ and $y := x$.
2. Independently for each $i \in \{0,1,\dots,n-1\}$
3.     With probability $p_{\text{aiNet}}(v) := (1/\rho)\cdot e^{-v}$ set $y[i] := 1 - y[i]$.
```

We restrict our attention to $\mu = 1$ using the normalization method described in Algorithm 4.4. This leads to the following refined mutation probability.

$$p_{\text{aiNet}}(\textsc{OneMax}(x)) = (1/\rho)\cdot e^{-\frac{\textsc{OneMax}(x)}{n}} \tag{4.7}$$

In the following, we consider similar settings for ρ as in the previous section.

Setting $\rho = 1$

We observe, that for $\rho = 1$ we have $p_{\text{aiNet}} = p_{\text{CLONALG}}$ and thus, we can immediately carry over the result from the previous section (see Theorem 4.6).

Corollary 4.23. *Let $\mu = 1$ and $\rho = 1$. The optimization time of Algorithm 3.1 using the mutation operator from Algorithm 4.3 and the normalization method from Algorithm 4.4 is*

$$T_{p_{aiNet},\textsc{OneMax}} \geq e^{c\cdot n}$$

($c > 0$ constant, sufficiently small) with probability at least $1 - e^{-\Omega(n)}$.

Setting $\rho = \Theta(n)$

For $\rho = \Theta(n)$ the mutation probability from eq. (4.7) becomes

$$p_{\text{aiNet}}(\textsc{OneMax}(x)) = \frac{1}{\Theta(n)} \cdot e^{-\frac{\textsc{OneMax}(x)}{n}} = \Theta\left(\frac{1}{n}\right)$$

and thus, we can adapt the analysis executed by Droste et al. (2002) in very much the same way as done in Theorem 4.2.

Theorem 4.24. *Let $\mu = 1$ and $\rho = \Theta(n)$. The expected optimization time of Algorithm 3.1 using the mutation operator from Algorithm 4.6 and the normalization method from Algorithm 4.4 on* OneMax *is*

$$E(T_{p_{aiNet},\textsc{OneMax}}) = \Theta(n \log n).$$

Proof. The proof is similar to that of Theorem 4.2. For the upper bound, we estimate the probability to increase the function value by at least one and apply fitness-layer arguments. For the lower bound, we adopt the analysis by Droste et al. (2002) for some mutation probability c/n ($c > 0$ constant). Since the concrete calculations differ only in the constants from those in Theorem 4.2, we omit the details here. $\square$

4.5. Experimental Supplements

In the preceding theoretical analysis we have seen results for different parameterizations of inversely fitness-proportional mutation probability, in particular with respect to polynomial and exponential optimization times. While the straightforward implementation (Algorithm 4.1) considered first as well as a proper parametrization of the opt-aiNet mutation operator (Algorithm 4.6) yield a polynomial optimization time (Theorem 4.2 and 4.24), we have an exponential lower bound (Theorem 4.3) for the Hamming distance based operator (Algorithm 4.2). Moreover, we have learned, that in case of the CLONALG operator (Algorithm 4.3) the normalization method makes a decisive difference (Section 4.3).

However, not all negative results, i. e., exponential lower bounds on the optimization time, are equal. For example, the CLONALG mutation operator with $\rho = \ln n$ in the (1+1) setting yields a smaller negative drift than the respective operator with $\rho = O(1)$ or the Hamming distance based mutation probability. Thus, the probability that the global optimum of OneMax is not found converges much more slowly to 1 than for the other variants. Moreover, the initial regions of the search space, where we were able to prove an expected polynomial optimization time, is largest for CLONALG and $\rho = \ln n$. Figure 4.2 visualizes this effect for different values of n, where $O(\log(n))$ corresponds to the Hamming distance based mutation probability and $O(n/\log n)$ belongs to the CLONALG mutation with $\rho = \ln n$. By Chernoff bounds, the initial bit string x_0 contains $n/2 \pm O(\sqrt{n})$ 1-bits with high probability. Numerical values for functions $O(f(n))$ are obtained by simply calculating $f(n)$.

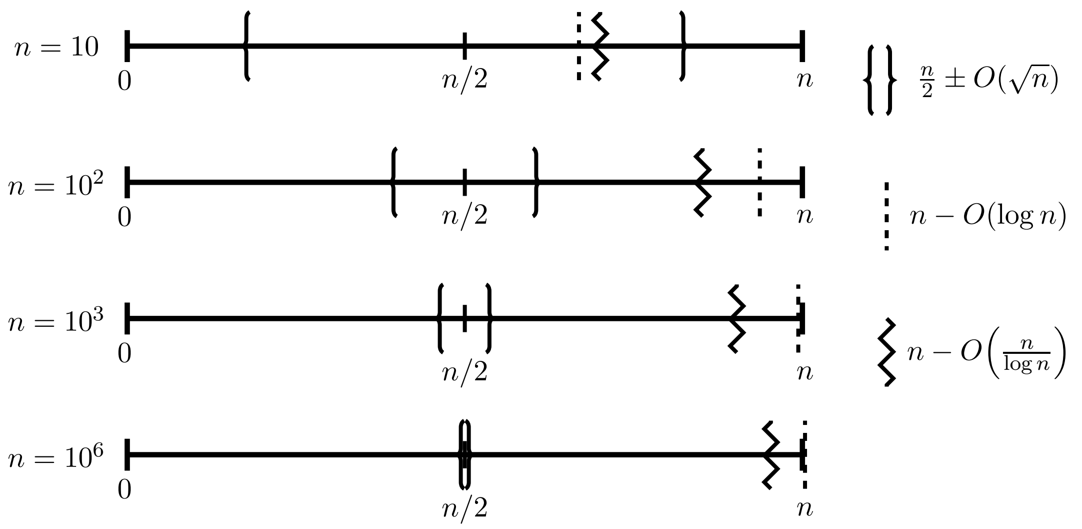


Figure 4.2.: The relative size of significant regions in the search space.

We recognize that for small values of n, we have $n/2 - \sqrt{n} < n/\log n < \log n$, whereas for large n we have the reverse order $\log n < n/\log n < n/2 - \sqrt{n}$. As our results are asymptotical, the behavior presented in the foregoing theorems rather corresponds to large n and thus, for n small enough the considered algorithm might deliver suitable performance. We further investigate this in the following. We start with an empirical comparison of different input sizes. Afterwards, we study the optimal choice of the parameter ρ for the CLONALG and opt-aiNet mutation operators. For the sake of comparison we visualize the mutation probabilities considered in this chapter in Figure 4.3.

4.5.1. Optimization Times of Different Mutation Operators

We first consider the optimization times of the different mutation operators considered in the preceding sections within the algorithmic framework and $\mu = 1$. For each of these operators, we perform 100 independent runs of Algorithm 3.1. For CLONALG we consider $\rho \in \{1, \ln n, n/e, n\}$, while for opt-aiNet $\rho \in \{1, n/e, n\}$ is used. Note, that for $\rho = 1$ the two operators are equal and thus, we only need to investigate this setting for one of the operators. The choice $\rho = n/e$ stems from the fact that for this value the mutation probability of opt-aiNet converges to $1/n$. Recall, that this is exactly the same motivation as for considering $\rho = \ln n$ for CLONALG. We remark that the mutation operators of CLONALG and opt-aiNet have been compared experimentally on OneMax for a bit string of length $n = 100$ (Cutello et al. 2005b).

We plot all results using box-and-whisker plots as defined in Definition B.1. Note, that we use logarithmic scales for the y-axis in all considered cases. Moreover, we use logarithmic scale for the x-axis when considering large values for n. For the purpose of comparison, we additionally consider the corresponding results for standard bit mutations.

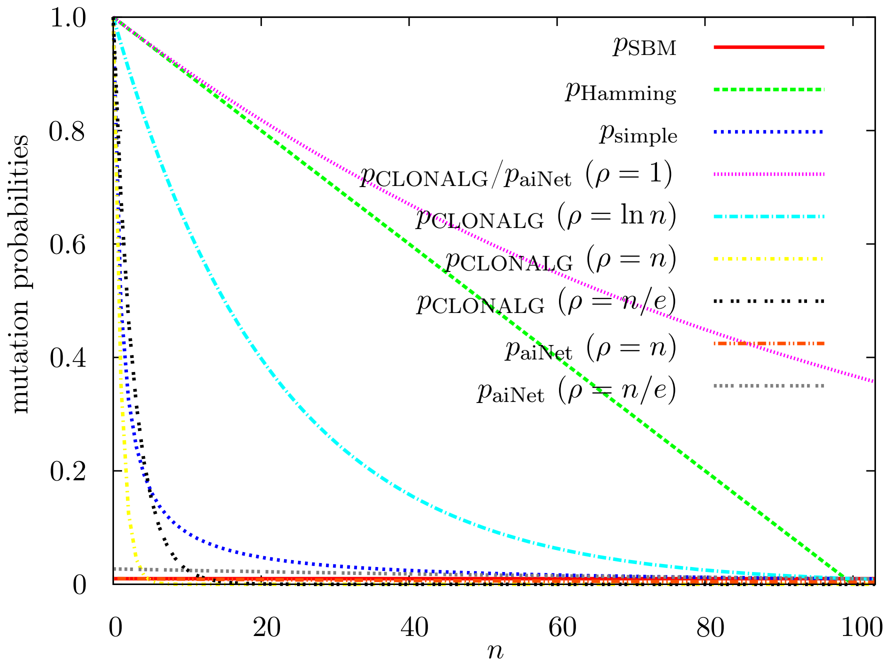


Figure 4.3.: Visualization of the mutation probabilities considered in this chapter.

The values of n used depend on the considered operator since the expected costs can, depending on the operator, become large already for not too large values of n. We present results for standard bit mutations in Figure 4.4 and for the two simple inversely fitness-proportional operators in Figure 4.5. As expected, standard bit mutations and the straightforward implementation from Algorithm 4.1 perform well and we consider quite large values for n, i. e., $n \in \{10, 20, \ldots, 100, 200, \ldots, 1000, \ldots, 10^6\}$. Both variants are able to optimize OneMax on 10^6 bits in less than 10^8 iterations. In contrast to that the Hamming distance based mutation operator performs badly, yielding results for $n \in \{10, 20, \ldots, 320\}$, only, where already for $n = 300$ we have more than 10^9 iterations.

The plots for CLONALG can be found in Figure 4.6. For CLONALG and $\rho = 1$ already $n = 50$ leads to approximately 10^{10} iterations. We therefore consider $n \in \{10, 20, \ldots, 50\}$ in this case. For $\rho = n$ the performance is even worse as around 10^{13} iterations are needed for $n = 30$, yielding $n \in \{10, 20, 30\}$. For $\rho = n/e$ we have 10^8 for $n = 50$ and again do experiments for $n \in \{10, 20, \ldots, 50\}$. However, $\rho = \ln n$ performs (despite the exponential optimization time with high probability) surprisingly well. We perform experiments for $n \in \{10, 20, \ldots, 100, 200, \ldots, 1000, \ldots, 10^6\}$ and observe, that for $n \leq 10^5$ there is hardly any significant difference between the results of standard bit mutations and CLONALG visible from the experiments. Only for large input sizes the difference in optimization time becomes obvious.

Finally, the results for opt-aiNet are presented in Figure 4.7. We see that, as expected, $\rho = n/e$ and $\rho = n$ both yield good performance and we again perform experience for $n \in \{10, 20, \ldots, 100, 200, \ldots, 1000, \ldots, 10^6\}$.

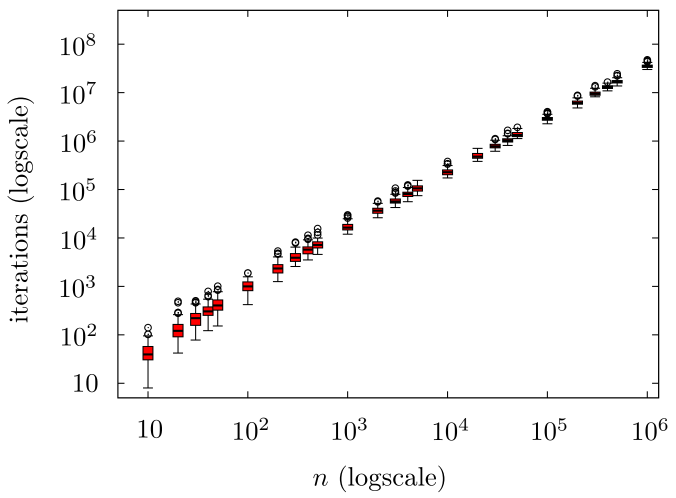
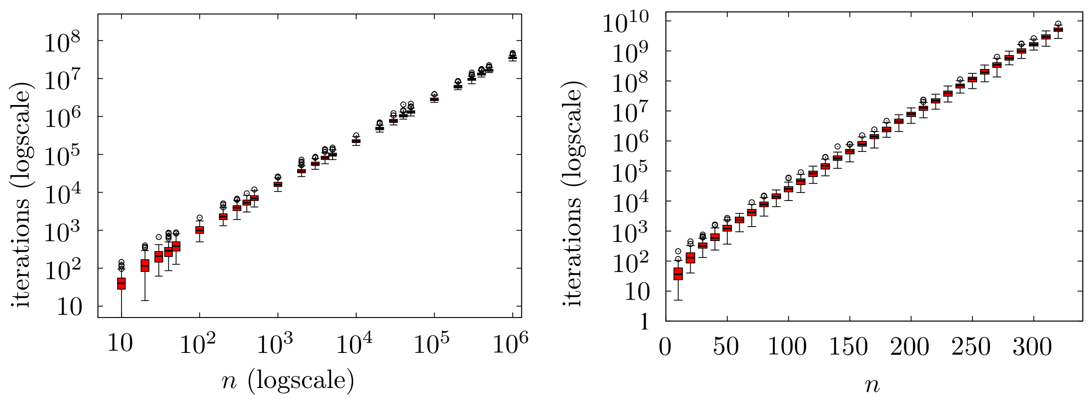


Figure 4.4.: Optimization time of Algorithm 3.1 using standard bit mutations (Algorithm 3.5), data from 100 runs.

For the purpose of comparison, we additionally plot the observed medians of all considered operators in one joint diagram (Figure 4.8). We see that, apart from CLONALG with $\rho = \ln n$ our expectations from the theoretical results are met. Moreover, the variants where we have proven an expected polynomial optimization time, all perform very similar. We investigate this further by only plotting the observed medians of the algorithms with good performance for $n \in \{10^5, 2 \cdot 10^5, \dots, 10^6\}$. (Figure 4.9).

We observe that the number of iterations of the two opt-aiNet variants differ by a constant factor. Moreover, for very large values of n it becomes obvious that the optimization time of CLONALG with $\rho = \ln n$ is more than a constant factor larger than for the other operators. For the other variants considered, there is hardly any difference visible in the plots. We conclude that standard bit mutations, the straightforward implementation and the mutation operator from opt-aiNet with $\rho = n/e$ perform pretty much identically when optimizing OneMax. Note, that this can be explained by the fact that in all these cases the mutation probability is $\Theta(1/n)$, at least in relevant parts of the search space. Moreover, from a practical point of view, CLONALG with $\rho = \ln n$ yields comparable performance if n is not too large. The operators not depicted in Figure 4.9 are unsuitable for the problem considered.

Finally, we perform experiments for the mutation operator from CLONALG for larger collection of search points. We consider $\mu = 2$ and $\mu = \sqrt{n}$ and consider $n \in \{10, 20, \dots, 100, 200, \dots, 1000, \dots, 5 \cdot 10^5\}$ and $n \in \{10, 20, \dots, 100, 200, \dots, 1000, \dots, 10^5\}$, respectively. For the purpose of comparison we present again the corresponding results for standard bit mutations. The results are depicted in Figure 4.10. We see that again, there is hardly any difference visible when comparing the performance of both operators.



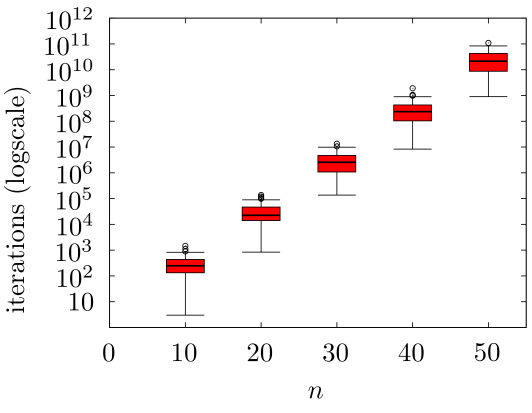
(a) p_{simple} (b) p_{Hamming}

Figure 4.5.: Optimization time of Algorithm 3.1 using the straightforward implementation (Algorithm 4.1) and the Hamming distance based mutation probability (Algorithm 4.2), data from 100 runs.

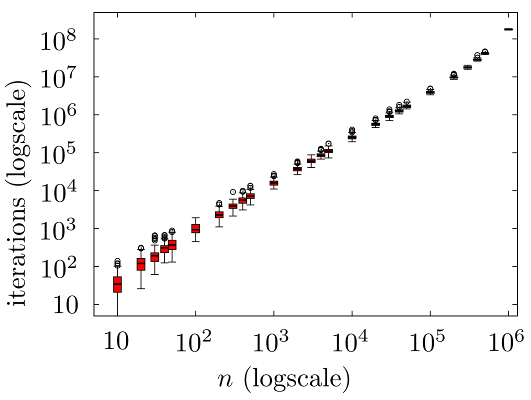
4.5.2. On the Optimal Choice for ρ

Considering our theoretical and experimental results OneMax, we speculate that for this function the 'optimal' setting for ρ in the mutation operators from CLONALG and opt-aiNet are approximately $\ln n$ and n/e, respectively. Since we do not have formal proofs for this conjecture, we perform experiments. To this end, we let Algorithm 3.1 with the two mutation operators run with different values of ρ and n. We fix the maximal number of iterations executed to the corresponding observed upper quartile from the experiments in the previous section (for $\rho = \ln n$ in the case of CLONALG and $\rho = n/e$ in the case of opt-aiNet). To be more precise, we perform experiments for $n \in \{10, 20, \ldots, 100, 200, \ldots, 1000, 2000, \ldots, 100000\}$ in both cases. For CLONALG we start with $\rho = \ln n$ and then successively increase (and accordingly decrease) ρ by 0.1 until the number of successes has reached 0 (for at least 5 consecutive values of ρ). Similarly, for opt-aiNet, we start with $\rho = n/e$ and then successively multiply ρ with a factor of 1.05 (or 0.95, respectively) until the same stopping criterion is fulfilled. We plot the number of successful trials in 100 independent runs in Figure 4.11. Moreover, we plot the functions $\ln n - 1$, $\ln n$, and $\ln n + 1$ for CLONALG and $n/(2e)$, n/e, and $2n/e$ for comparison.

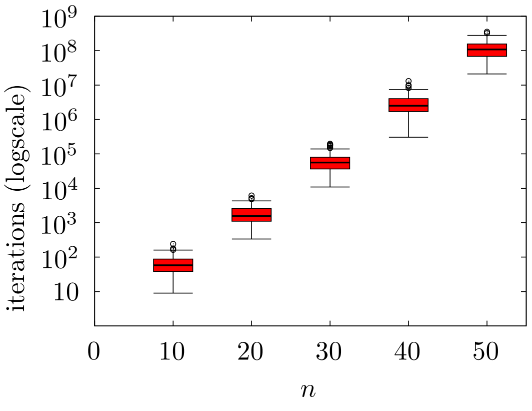
We see that our speculation is supported by the result of our experiments, in particular for opt-aiNet. For CLONALG a value slightly larger than $\ln n$ seems to be better. However, decreasing ρ below $\ln n$ seems to be a particularly bad idea. Moreover, we recognize that the mutation operator from opt-aiNet is much more robust with respect to the choice of ρ. While a small change might have a huge negative effect when using the CLONALG mutation probability, there is a quite large range of values around n/e that yields similar results. However, this observation is not too surprising when looking at the mutation operators. For CLONALG the parameter ρ is situated in the exponent



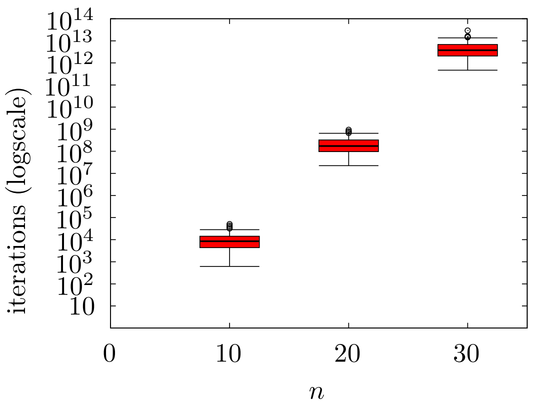
(a) $\rho = 1$



(b) $\rho = \ln n$



(c) $\rho = n/e$



(d) $\rho = n$

Figure 4.6.: Optimization time of Algorithm 3.1 using the mutation operator from CLONALG (Algorithm 4.3) with different values for ρ and the normalization method from Algorithm 4.4, data from 100 runs.

of the inverse exponential function. In contrast to that, in opt-aiNet we divide the result of this inverse exponential function by ρ. Thus, it is somehow obvious that the choice of ρ is much more crucial when considering CLONALG.

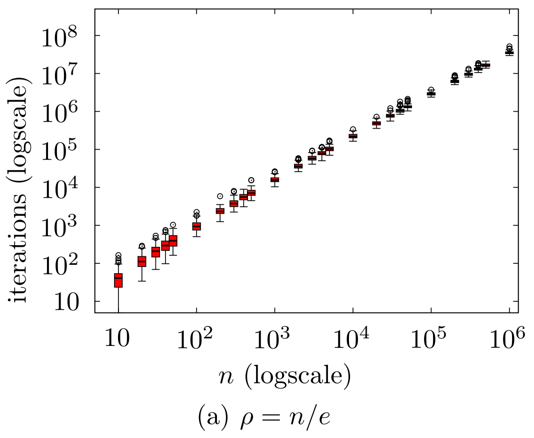


Figure 4.7.: Optimization time of Algorithm 3.1 using the mutation operator from opt-aiNet (Algorithm 4.6) with different values for ρ and the normalization method from Algorithm 4.4, data from 100 runs.

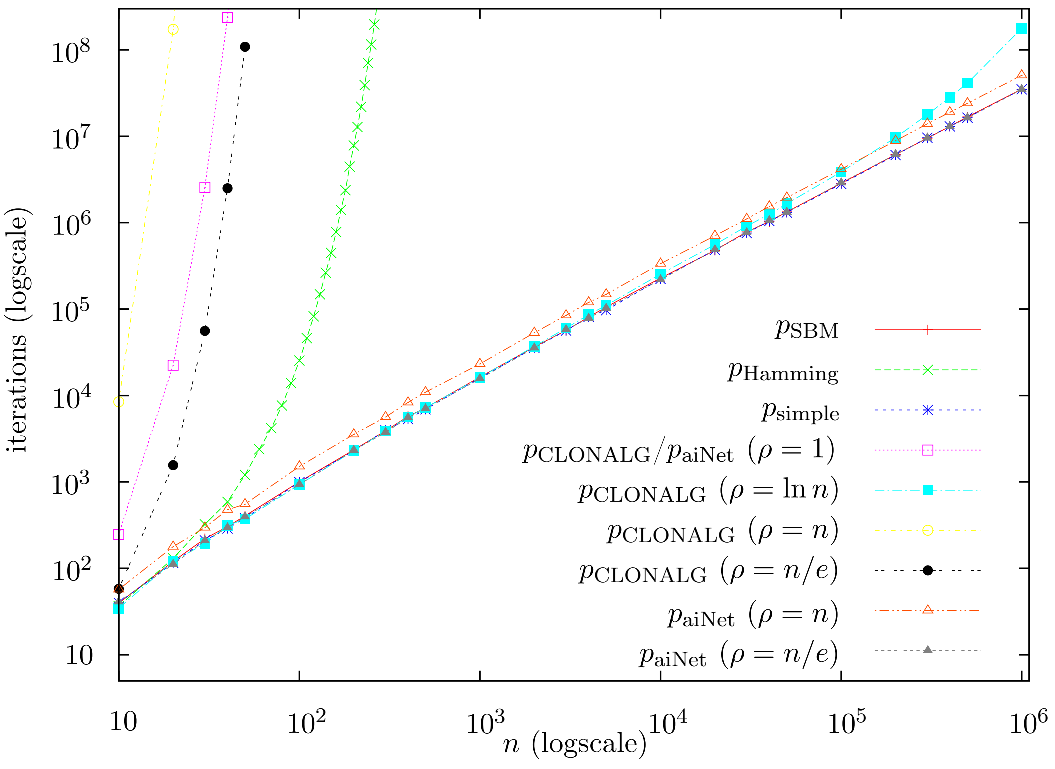


Figure 4.8.: Median optimization times of Algorithm 3.1 using different mutation operators, data from 100 runs.

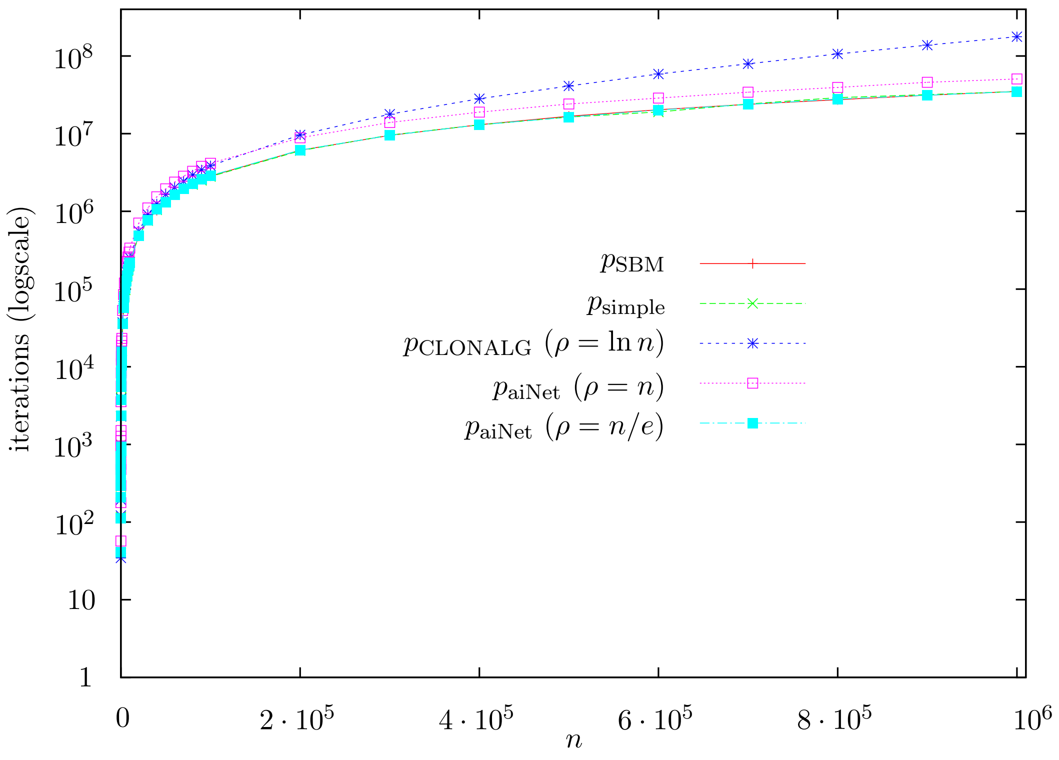
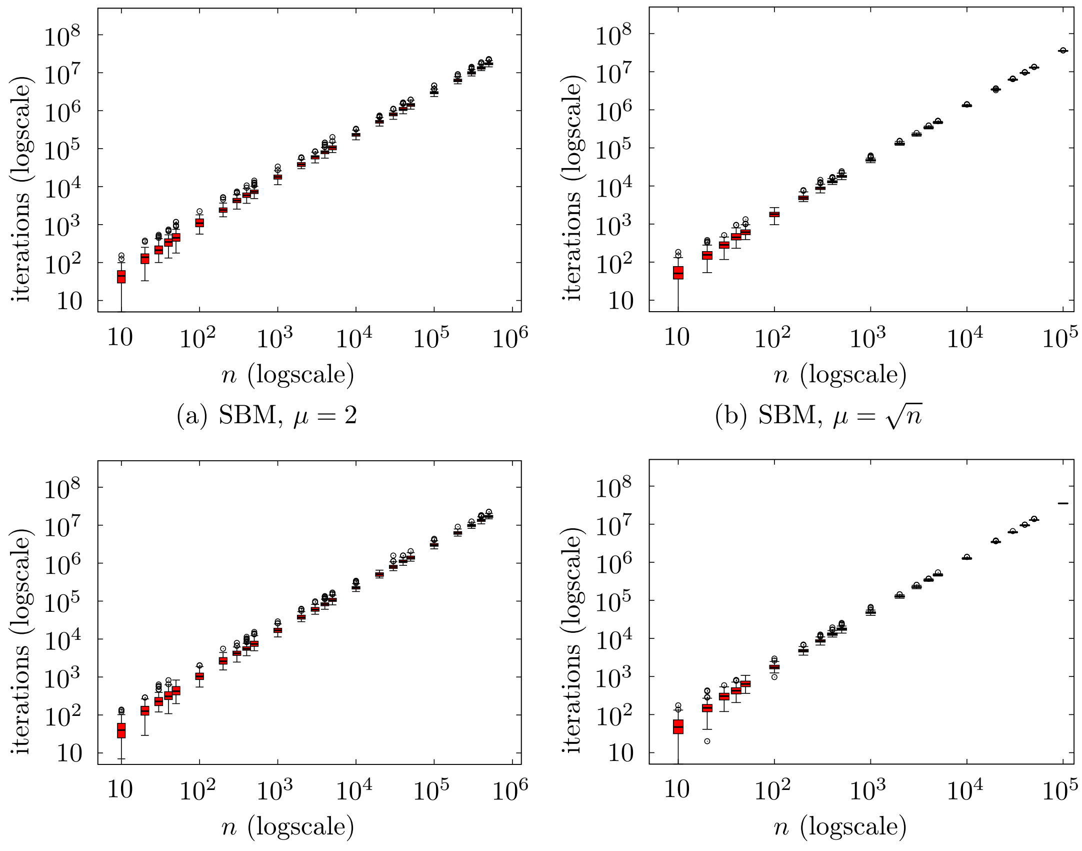


Figure 4.9.: Median optimization times of Algorithm 3.1 using different mutation operators, data from 100 runs.



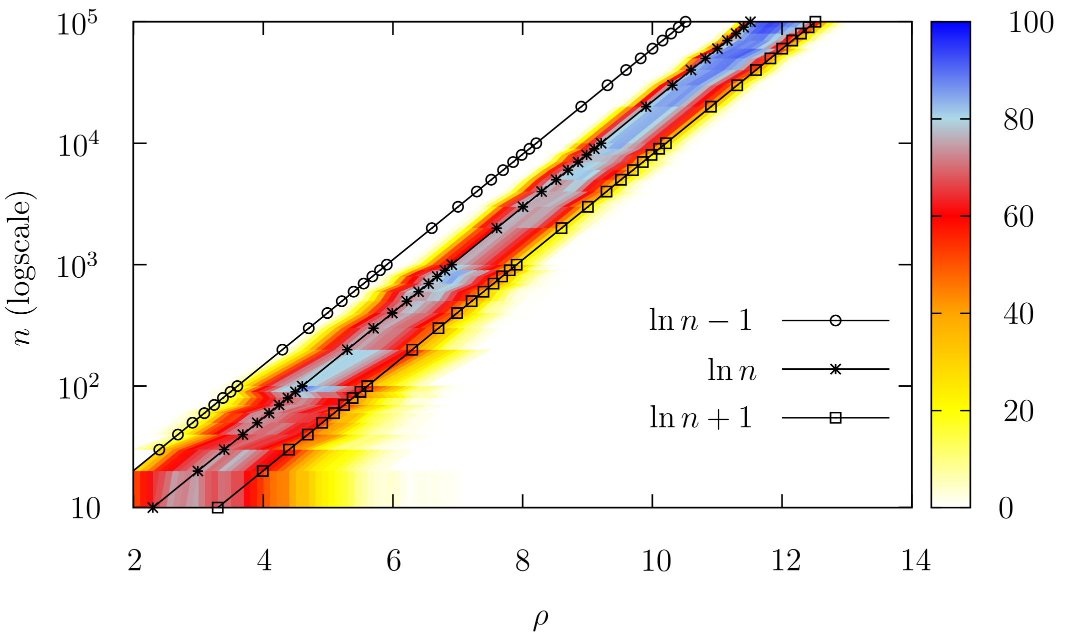
(a) SBM, $\mu = 2$

(b) SBM, $\mu = \sqrt{n}$

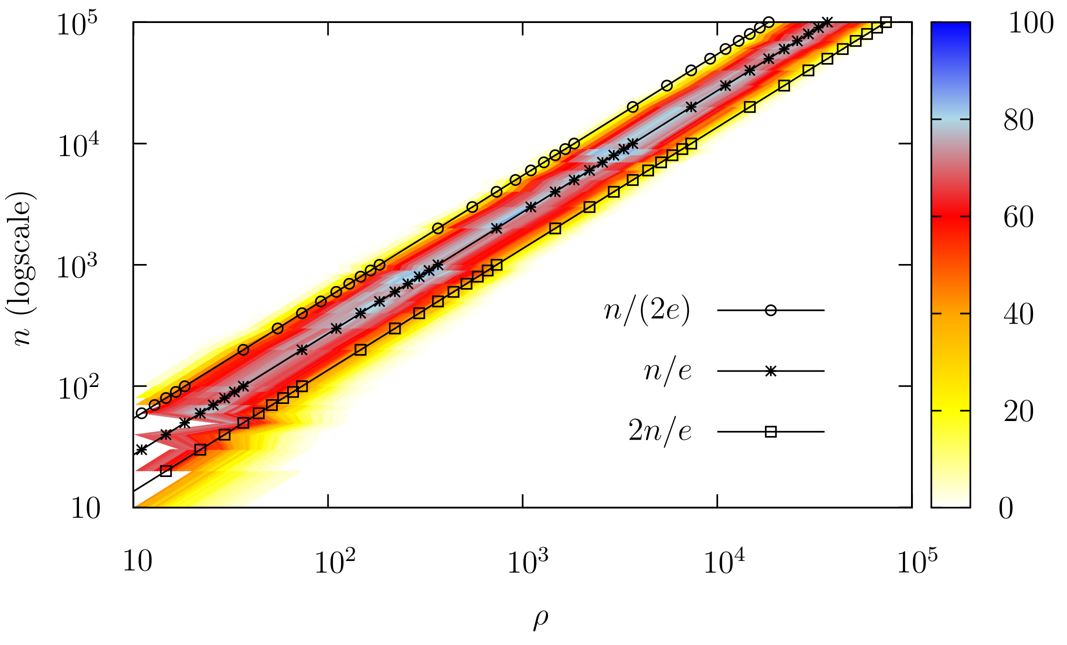
(c) p_{CLONALG}, $\mu = 2$

(d) p_{CLONALG}, $\mu = \sqrt{n}$

Figure 4.10.: Optimization time of Algorithm 3.1 using standard bit mutation (Algorithm 3.5) or the mutation operator from CLONALG (Algorithm 4.3) with $\rho = \ln n$ and the normalization method from Algorithm 4.5, data from 100 runs.



(a) CLONALG



(b) opt-aiNet

Figure 4.11.: Success rates in 100 runs of Algorithm 3.1 with the CLONALG (Algorithm 4.3) and opt-aiNet mutation operator (Algorithm 4.6) using the normalization method from Algorithm 4.4 and different values for ρ.

5. Contiguous Hypermutations

In this chapter, we investigate the performance of somatic contiguous hypermutations, the main characteristic of the B-Cell algorithm (Kelsey and Timmis 2003), in the same spirit as done in the previous chapter for inversely fitness-proportional mutation probabilities. We embed the mutation operator into our algorithmic framework (Algorithm 3.1) and present results on well-known and instructive example functions in order to get an understanding of assets and drawbacks of this kind of mutation operator. We show that while there are serious limitations to the performance of this type of operator even for simple optimization tasks, for some types of optimization problems it performs much better than standard bit mutations most often used in evolutionary algorithms.

Somatic contiguous hypermutations as introduced by Kelsey and Timmis (2003) and later formally described by Clark et al. (2005) follow the idea to decide randomly about a contiguous region of the search point and flip all bits within this region with probability r. They do not change any bit that is outside of this region. To be more precise, the region is selected by randomly deciding on a starting position p (called *hotspot*) and a length l of the interval to be mutated. Note, that the bit string is not assumed to be cyclic and therefore the contiguous region does not wrap around (Clark 2008). The concrete implementation can be found in Algorithm 5.1.

Algorithm 5.1 Somatic Contiguous Hypermutations CHM_1 (Clark et al. 2005).

```
FUNCTION mutate(x):
1. Select $p \in \{0, 1, \ldots, n-1\}$ uniformly at random.
2. Select $l \in \{0, 1, \ldots, n\}$ uniformly at random.
3. For $i := 0$ to $\min\{l-1, n-1-p\}$ do
4.     With probability $r$ set $x[p+i] := 1 - x[p+i]$.
```

We use the original mutation operator as an inspiration and consider not only the concrete operator due to Clark et al. (2005) but also two variants that can be considered concrete examples for somatic contiguous hypermutation operators. We show that the three instantiations behave quite differently in general.

The first variant (Jansen and Zarges 2009a) is motivated by the strong positional bias and the strongly different mutation probabilities for mutations of single bits depending on their location. Without additional knowledge about the roles of different bits any such bias is undesirable (Droste and Wiesmann 2003). Instead of choosing a hotspot and an interval length, two positions p_1 and p_2 are chosen randomly and all bits between these two positions are flipped with probability r. However, the resulting operator still suffers from a (different) positional bias. The important difference is that for 1-bit mutations the probability is independent of the position. Since 1-bit mutations can be crucial this kind ob being bias-free is important. We define the operator in Algorithm 5.2.

Algorithm 5.2 Somatic Contiguous Hypermutations CHM$_2$ (Jansen and Zarges 2009a).

```
FUNCTION mutate(x):
1. Select $p_1 \in \{0, 1, \ldots, n-1\}$ uniformly at random.
2. Select $p_2 \in \{0, 1, \ldots, n-1\}$ uniformly at random.
3. For $i := \min\{p_1, p_2\}$ to $\max\{p_1, p_2\}$ do
4.     With probability $r$ set $x[i] := 1 - x[i]$.
```

The positional bias of CHM$_2$ motivates the second variant that is essentially the original operator, but wrapping around. This last variant has no positional bias at all. We come back to this point later when discussing properties of the different operators. Note, that it is conceivable that Kelsey and Timmis (2003) had this last version in mind when introducing the B-Cell algorithm. However, there definition is not explicit. We define the resulting mutation operators formally in Algorithm 5.3.

Algorithm 5.3 Somatic Contiguous Hypermutations CHM$_3$ Wrapping Around (Jansen and Zarges 2011a).

```
FUNCTION mutate(x):
1. Select $p \in \{0, 1, \ldots, n-1\}$ uniformly at random.
2. Select $l \in \{0, 1, \ldots, n\}$ uniformly at random.
3. For $i := 0$ to $l - 1$ do
4.     With probability $r$ set $x[(p+i) \bmod n] := 1 - x[(p+i) \bmod n]$.
```

In the following, we first derive some properties of the considered mutation operators and present general bounds on the optimization time of Algorithm 3.1 using these operators. Afterwards, we investigate the performance on several concrete example functions from the literature. We close with empirical results that complement the analytical part by further investigating stated conjectures that could not be proven. The results in this chapter are based on the work done in Jansen and Zarges (2009a, 2011a).

5.1. Properties of Different Variants of Contiguous Hypermutations

The three variants of somatic contiguous hypermutations differ considerably in the probability of mutations. Thus, we first analyze these important properties of the mutation operators. We consider two different events and compare them across the three somatic contiguous hypermutation operators. On one hand, we consider for $i \in \{0, 1, \ldots, n-1\}$ the probability of the i-th bit to be mutated in a single mutation. On the other hand, we consider for $i \in \{0, 1, \ldots, n-1\}$ the probability of the i-th bit to be the only bit that is mutated in a single mutation. Note that such single bit mutations are often important for locating an optimum of an objective function exactly. Finally, we compare the expected number of bits to be mutated in a single mutation.

Lemma 5.1 (Jansen and Zarges (2011a)). *Let $n \in \mathbb{N}$, $i \in \{0, 1, \dots, n-1\}$ and $0 < r \leq 1$.*

$$\textit{Prob}\,(x[i] \textit{ mutated by } \textit{CHM}_1) = r \cdot \frac{(2n-i)(i+1)}{2n(n+1)} \tag{5.1}$$

$$\textit{Prob}\,(x[i] \textit{ mutated by } \textit{CHM}_2) = r \cdot \frac{1}{n} + \frac{2i(n-1-i)}{n^2} \tag{5.2}$$

$$\textit{Prob}\,(x[i] \textit{ mutated by } \textit{CHM}_3) = r \cdot \frac{1}{2} \tag{5.3}$$

Proof. In all three equations, we see the factor r stemming from the probability r that a bit within the contiguous region that is selected randomly is actually mutated. For CHM_1 and CHM_3 we have a factor of $1/n$ for choosing a specific value of p and a factor $1/(n+1)$ for choosing a specific value of l. For CHM_2 we have a factor of $1/n^2$ for choosing specific values of p_1 and p_2.

For CHM_1 the i-th bit is mutated if $p \leq i$ and $p - 1 + l \geq i$ both hold. For $j \in \{0, 1, \dots, i\}$ we have $\text{Prob}\,(p = j) = 1/n$ and $\text{Prob}\,(j - 1 + l \geq i) = \text{Prob}\,(l \geq i + 1 - j)$ $= (n + j - i)/(n+1)$. Using Lemma B.19 this yields

$$\begin{aligned}
&\text{Prob}\,(x[i] \text{ is mutated by } \text{CHM}_1) = r \cdot \sum_{j=0}^{i} \frac{1}{n} \cdot \frac{n+j-i}{n+1} \\
&= r \cdot \frac{1}{n(n+1)} \left((i+1)(n-i) + \sum_{j=0}^{i} j \right) = r \cdot \frac{1}{n(n+1)} \left((i+1)(n-i) + \frac{i(i+1)}{2} \right) \\
&= r \cdot \frac{(2n-i)(i+1)}{2n(n+1)}
\end{aligned}$$

and proves eq. (5.1).

For CHM_2 the i-th bit is mutated if $p_1 \leq i$ and $p_2 > i$, $p_2 < i$ and $p_1 \geq i$ or $p_1 = p_2 = i$. Thus, we have

$$\begin{aligned}
\text{Prob}\,(x[i] \text{ is mutated by } \text{CHM}_2) &= r \cdot \left(\frac{i+1}{n} \cdot \frac{n-i-1}{n} + \frac{i}{n} \cdot \frac{n-i}{n} + \frac{1}{n^2} \right) \\
&= r \cdot \frac{in - i^2 - i + n - i - 1 + in - i^2 + 1}{n^2} = r \cdot \left(\frac{1}{n} + \frac{2i(n-1-i)}{n^2} \right)
\end{aligned}$$

proving eq. (5.2).

For CHM_3 we observe that due to the wrapping around the probabilities are equal for all i. We thus consider the situation only for $i = n-1$ where things are less complicated since the starting position p of the contiguous region is never to the right of i. For each position $p = n - 1 - j$ ($j \in \{0, 1, \dots, n-1\}$) there are exactly $n - j$ values for l such

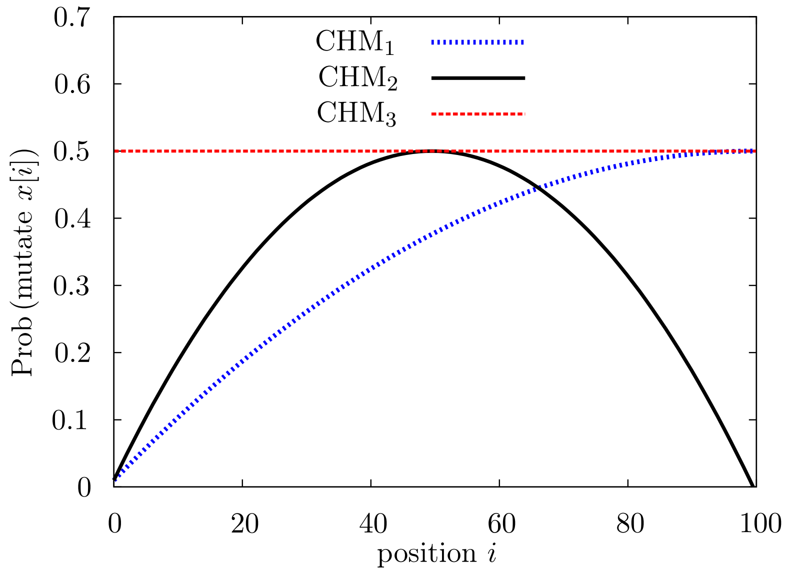


Figure 5.1.: Probability for mutating bit $x[i]$ ($i \in \{0, 1, \ldots, n-1\}$) in a single mutation using somatic contiguous hypermutation CHM$_1$, CHM$_2$, and CHM$_3$, respectively, each with $r = 1$ and $n = 100$ (Jansen and Zarges 2011a).

that $x[i] = x[n-1]$ is mutated. Thus we have (using Lemma B.19)

$$\text{Prob}\,(x[i] \text{ is mutated by CHM}_3) = r \cdot \sum_{j=0}^{n-1} \frac{1}{n} \cdot \frac{n-j}{n+1} = r \cdot \frac{1}{n(n+1)} \sum_{j=1}^{n} j = r \cdot \frac{1}{2}$$

proving eq. (5.3). □

We give a graphical representation of the probability for mutating bit $x[i]$ and the special case $r = 1$ in Figure 5.1. We see that only CHM$_3$ has no positional bias. Both, CHM$_1$ and CHM$_2$ have a strong positional bias with respect to the probability to mutate some bit. For CHM$_1$ this probability is strictly increasing with i, for CHM$_2$ it is equal for i and $(n-1)-i$ and thus symmetric. Moreover, CHM$_1$ is also biased with respect to 1-bit mutations having a much larger probability for such a mutation at position $n-1$ than for any other position. We investigate this in the next lemma by considering the probability for a specific 1-bit mutation.

Lemma 5.2 (Jansen and Zarges (2011a)). *Let $n \in \mathbb{N}$, $i \in \{0, 1, \ldots, n-1\}$ and $0 < r \leq 1$. For $0 < r < 1$*

$$\textit{Prob}\,(\textit{only } x[i] \textit{ mutated by CHM}_1) = \frac{i(1-r)^{n-i-1}}{n(n+1)} + \frac{(1-r)^{n+1} - 2(1-r)^{n-i} - (1-r)^{i+1} + 1 + (1-r)^n}{rn(n+1)} \tag{5.4}$$

$$\mathit{Prob}\left(\mathit{only}\ x[i]\ \mathit{mutated\ by}\ \mathit{CHM}_2\right) = \frac{r}{n^2} + 2\sum_{p_1=0}^{i}\ \sum_{p_2=\max\{i,p_1+1\}}^{n-1} \frac{r(1-r)^{p_2-p_1}}{n^2} \tag{5.5}$$

$$\mathit{Prob}\left(\mathit{only}\ x[i]\ \mathit{mutated\ by}\ \mathit{CHM}_3\right) = \frac{1-(1-r)^n}{rn(n+1)} - \frac{(1-r)^n}{n+1} \tag{5.6}$$

holds. In the case $r = 1$, the following holds.

$$\mathit{Prob}\left(\mathit{only}\ x[i]\ \mathit{mutated\ by}\ \mathit{CHM}_1\right) = \begin{cases} \frac{1}{n(n+1)} & \mathit{if}\ i < n-1 \\ \frac{1}{n+1} & \mathit{otherwise} \end{cases} \tag{5.7}$$

$$\mathit{Prob}\left(\mathit{only}\ x[i]\ \mathit{mutated\ by}\ \mathit{CHM}_2\right) = \frac{1}{n^2} \tag{5.8}$$

$$\mathit{Prob}\left(\mathit{only}\ x[i]\ \mathit{mutated\ by}\ \mathit{CHM}_3\right) = \frac{1}{n(n+1)} \tag{5.9}$$

Proof. For the proof of eq. (5.4) we consider the definition of CHM$_1$ and see that

$$\text{Prob}\left(\text{only } x[i] \text{ mutated by CHM}_1\right) = \left(\sum_{p=0}^{i}\ \sum_{l=i-p+1}^{n-p} \frac{r(1-r)^{l-1}}{n(n+1)}\right) + \left(\sum_{p=0}^{i} p \cdot \frac{r(1-r)^{n-p-1}}{n(n+1)}\right)$$

holds. The first sum accounts for all cases where a position p with $p \leq i$ is selected and the length of the interval is $i-p+1 \leq l \leq n-p$ so that the last position of the mutated region is at most $n-1$ and thus within the bit string x. In this case $x[i]$ needs to flip and the other $l-1$ bits remain unchanged with probability $(1-r)^{l-1}$. For $l > n-p$ the mutated region extends beyond the bit string so that only $n-p-1$ bits remain unchanged. It is a tedious but not difficult exercise to see that this equals eq. (5.5) for $0 < r < 1$. Note, that the main tool used in the calculations below is the closed form of the geometric series (see Lemma B.7).

$$\begin{aligned}
&\left(\sum_{p=0}^{i}\ \sum_{l=i-p+1}^{n-p} \frac{r(1-r)^{l-1}}{n(n+1)}\right) + \left(\sum_{p=0}^{i} p \cdot \frac{r(1-r)^{n-p-1}}{n(n+1)}\right) \\
=&\frac{r}{n(n+1)}\left(\left(\sum_{p=0}^{i}\ \sum_{l=i-p+1}^{n-p} (1-r)^{l-1}\right) + \left(\sum_{p=0}^{i} p \cdot (1-r)^{n-p-1}\right)\right) \\
=&\frac{r}{n(n+1)}\left(\left(\sum_{p=0}^{i}\ \sum_{l=i-p}^{n-p-1} (1-r)^{l}\right) + \left(\sum_{p=1}^{i} p \cdot (1-r)^{n-p-1}\right)\right) \\
=&\frac{r}{n(n+1)}\left(\left(\sum_{p=0}^{i} \frac{(1-r)^{i-p} - (1-r)^{n-p}}{r}\right) + \left(\sum_{p=1}^{i} p \cdot (1-r)^{n-p-1}\right)\right)
\end{aligned}$$

$$
\begin{aligned}
=&\frac{1}{n(n+1)}\left(\left((1-r)^i\sum_{p=0}^{i}\left(\frac{1}{1-r}\right)^p\right)-\left((1-r)^n\sum_{p=0}^{i}\left(\frac{1}{1-r}\right)^p\right)\right)\\
&+\frac{r}{n(n+1)}\left(\sum_{p=1}^{i}p\cdot(1-r)^{n-p-1}\right)\\
=&\frac{1}{n(n+1)}\left(\left((1-r)^i\frac{1-\left(\frac{1}{1-r}\right)^{i+1}}{1-\frac{1}{1-r}}\right)-\left((1-r)^n\frac{1-\left(\frac{1}{1-r}\right)^{i+1}}{1-\frac{1}{1-r}}\right)\right)\\
&+\frac{r}{n(n+1)}\left(\sum_{p=1}^{i}p\cdot(1-r)^{n-p-1}\right)\\
=&\frac{1}{n(n+1)}\left(\left((1-r)^i\frac{1-r-\left(\frac{1}{1-r}\right)^{i}}{-r}\right)-\left((1-r)^n\frac{1-r-\left(\frac{1}{1-r}\right)^{i}}{-r}\right)\right)\\
&+\frac{r}{n(n+1)}\left(\sum_{p=1}^{i}p\cdot(1-r)^{n-p-1}\right)\\
=&\frac{1}{rn(n+1)}\left(\left((1-r)^n\left(1-r-\left(\frac{1}{1-r}\right)^{i}\right)\right)-\left((1-r)^i\left(1-r-\left(\frac{1}{1-r}\right)^{i}\right)\right)\right)\\
&+\frac{r}{n(n+1)}\left(\sum_{p=1}^{i}p\cdot(1-r)^{n-p-1}\right)\\
=&\frac{1}{rn(n+1)}\left((1-r)^{n+1}-(1-r)^{n-i}-\left((1-r)^{i+1}-1\right)\right)\\
&+\frac{r}{n(n+1)}\left(\sum_{p=1}^{i}p\cdot(1-r)^{n-p-1}\right)\\
=&\frac{(1-r)^{n+1}-(1-r)^{n-i}-(1-r)^{i+1}+1}{rn(n+1)}+\frac{r}{n(n+1)}\left(\sum_{p=1}^{i}\sum_{j=p}^{i}(1-r)^{n-j-1}\right)\\
=&\frac{(1-r)^{n+1}-(1-r)^{n-i}-(1-r)^{i+1}+1}{rn(n+1)}+\frac{r(1-r)^{n-1}}{n(n+1)}\left(\sum_{p=1}^{i}\sum_{j=p}^{i}\left(\frac{1}{1-r}\right)^j\right)\\
=&\frac{(1-r)^{n+1}-(1-r)^{n-i}-(1-r)^{i+1}+1}{rn(n+1)}+\frac{r(1-r)^{n-1}}{n(n+1)}\left(\sum_{p=1}^{i}\frac{\left(\frac{1}{1-r}\right)^p-\left(\frac{1}{1-r}\right)^{i+1}}{1-\frac{1}{1-r}}\right)
\end{aligned}
$$

$$
\begin{aligned}
=&\frac{(1-r)^{n+1}-(1-r)^{n-i}-(1-r)^{i+1}+1}{rn(n+1)}+\frac{r(1-r)^{n-1}}{n(n+1)}\left(\sum_{p=1}^{i}\frac{\left(\frac{1}{1-r}\right)^{i}-\left(\frac{1}{1-r}\right)^{p-1}}{r}\right)\\
=&\frac{(1-r)^{n+1}-(1-r)^{n-i}-(1-r)^{i+1}+1}{rn(n+1)}\\
&+\frac{(1-r)^{n-1}}{n(n+1)}\left(i\left(\frac{1}{1-r}\right)^{i}-\sum_{p=0}^{i-1}\left(\frac{1}{1-r}\right)^{p}\right)\\
=&\frac{(1-r)^{n+1}-(1-r)^{n-i}-(1-r)^{i+1}+1}{rn(n+1)}\\
&+\frac{(1-r)^{n-1}}{n(n+1)}\left(i\left(\frac{1}{1-r}\right)^{i}-\frac{1-\left(\frac{1}{1-r}\right)^{i}}{1-\frac{1}{1-r}}\right)\\
=&\frac{(1-r)^{n+1}-(1-r)^{n-i}-(1-r)^{i+1}+1}{rn(n+1)}\\
&+\frac{(1-r)^{n-1}}{n(n+1)}\left(i\left(\frac{1}{1-r}\right)^{i}-\frac{1-r-\left(\frac{1}{1-r}\right)^{i-1}}{r}\right)\\
=&\frac{(1-r)^{n+1}-2(1-r)^{n-i}-(1-r)^{i+1}+1+(1-r)^{n}}{rn(n+1)}+\frac{i(1-r)^{n-i-1}}{n(n+1)}
\end{aligned}
$$

For $r=1$ we need to have $p=i$ and $l=1$ if $i<n-1$, and $p=i$ and $l\neq 0$ if $i=n-1$. Thus, eq. (5.7) results as a special case.

The proof of eq. (5.5) follows quite directly from the definition of CHM$_2$. In

$$
\frac{r}{n^2}+2\sum_{p_1=0}^{i}\sum_{p_2=\max\{i,p_1+1\}}^{n-1}\frac{r(1-r)^{p_2-p_1}}{n^2}
$$

the term r/n^2 takes care of the special case $p_1=p_2$. If we have $p_1\neq p_2$ the cases $p_1<p_2$ and $p_1>p_2$ have equal contribution and lead to the factor 2. We consider the case $p_1<p_2$ and sum over all possible cases directly obtaining the claimed sum. For $r=1$ only $p_1=p_2$ needs to be considered implying eq. (5.8).

For the proof of eq. (5.6) we consider the definition of CHM$_3$ and see that

$$
\text{Prob}\left(\text{only } x[i] \text{ mutated by CHM}_3\right)=\sum_{d=0}^{n-1}\sum_{l=d+1}^{n}\frac{r(1-r)^{l-1}}{n(n+1)}
$$

holds in the following way. For some fixed position i we sum over all positions $p=(i-d)\bmod n$ with $d\in\{0,1,\ldots,n-1\}$. Given this position we have to have $l\geq d+1$, otherwise the bit $x[i]$ is not in the mutated region. Given a length l of this region we have that only $x[i]$ flips and the other $l-1$ bits remain unchanged with probability $r(1-r)^{l-1}$. Again making use of the geometric series (Lemma B.7), we see that

$$\sum_{d=0}^{n-1}\sum_{l=d+1}^{n}\frac{r(1-r)^{l-1}}{n(n+1)} = \frac{r}{n(n+1)}\sum_{d=0}^{n-1}\sum_{l=d}^{n-1}(1-r)^l$$
$$= \frac{1}{n(n+1)}\sum_{d=0}^{n-1}\Big((1-r)^d-(1-r)^n\Big) = \frac{1-(1-r)^n}{rn(n+1)} - \frac{(1-r)^n}{n+1}$$

holds and we have eq. (5.6). For $r=1$ only the case $d=0$, $l=1$ (or in other words $p=i$ and $l=1$) needs to be taken into account leading to eq. (5.9). □

Finally, the expected number of bits in an iteration can easily be derived using the results from Lemma 5.1. We see that with respect to the expected number of mutating bits all three somatic contiguous hypermutation operators are similar, mutating on average $\Theta(rn)$ bits, $r\cdot n/2$ for CHM$_3$ and almost exactly $r\cdot n/3$ for both, CHM$_1$ and CHM$_2$.

Lemma 5.3 (Jansen and Zarges (2011a)). *Let* $n\in\mathbb{N}$, $i\in\{0,1,\dots,n-1\}$ *and* $0<r\le 1$.

$$E(\#\textit{bits mutated by CHM}_1) = r\cdot\left(\frac{n}{3}+\frac{1}{6}\right) \tag{5.10}$$
$$E(\#\textit{bits mutated by CHM}_2) = r\cdot\left(\frac{n}{3}+\frac{2}{3n}\right) \tag{5.11}$$
$$E(\#\textit{bits mutated by CHM}_3) = r\cdot\frac{n}{2} \tag{5.12}$$

Proof. Again, the factor r in all equations stems from the probability r that a bit within the contiguous region that is selected randomly is actually mutated. The claimed expected values can simply be derived by using the results from Lemma 5.1 and the definition of the expected value (Definition B.4).

We obtain eq. (5.10) using eq. (5.1) and Lemma B.19 by

$$\sum_{i=0}^{n-1}\text{Prob}\,(x[i] \text{ is mutated by CHM}_1)$$
$$= r\cdot\sum_{i=0}^{n-1}\frac{(2n-i)(i+1)}{2n(n+1)} = r\cdot\frac{1}{2n(n+1)}\cdot\sum_{i=0}^{n-1}(2ni+2n-i^2-i)$$
$$= r\cdot\frac{1}{2n(n+1)}\cdot\left(2n\cdot\frac{n(n-1)}{2}+2n^2-\frac{n(n-1)(2n-1)}{6}-\frac{n(n-1)}{2}\right)$$
$$= r\cdot\frac{1}{2(n+1)}\cdot\left(n^2-n+2n-\frac{2n^2-3n+1}{6}-\frac{n-1}{2}\right)$$
$$= r\cdot\frac{2n^2+3n+1}{6(n+1)} = r\cdot\frac{(n+1)(4n+2)}{6(n+1)} = r\cdot\left(\frac{n}{3}+\frac{1}{6}\right).$$

Similarly, we obtain eq. (5.11) using eq. (5.2) and Lemma B.19 by

$$\begin{aligned}
&\sum_{i=0}^{n-1} \text{Prob}\left(x[i] \text{ is mutated by CHM}_2\right)\\
&= r \cdot \sum_{i=0}^{n-1} \left(\frac{1}{n} + \frac{2i(n-1-i)}{n^2}\right) = r \cdot \left(1 + \frac{2}{n^2}\sum_{i=0}^{n-1}(in - i - i^2)\right)\\
&= r \cdot \left(1 + \frac{2}{n^2} \cdot \left(\frac{n^2(n-1)}{2} - \frac{n(n-1)}{2} - \frac{n(n-1)(2n-1)}{6}\right)\right)\\
&= r \cdot \left(1 + \frac{1}{n} \cdot \left(n^2 - n - (n-1) - \frac{2n^2 - 3n + 1}{3}\right)\right)\\
&= r \cdot \left(1 + \frac{n^2 - 3n + 2}{3n}\right) = r \cdot \left(\frac{n^2+2}{3n}\right) = r \cdot \left(\frac{n}{3} + \frac{2}{3n}\right).
\end{aligned}$$

Finally, eq. (5.12) follows as an immediate consequence of eq. (5.3). □

We intend to compare the results of contiguous hypermutations with the respective results of standard bit mutations with mutation probability $1/n$ (SBM, see Algorithm 3.5). For this mutation operator it is easy to show the above properties. We formalize this in the following lemma. Note, that all equations follow directly from the definition of standard bit mutations.

Lemma 5.4 (Jansen and Zarges (2011a)). *Let $n \in \mathbb{N}$ and $i \in \{0, 1, \ldots, n-1\}$.*

$$\begin{aligned}
\textit{Prob}\left(\textit{only } x[i] \textit{ is mutated by SBM}\right) &= \frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-1} = \Theta\left(\frac{1}{n}\right)\\
\textit{Prob}\left(x[i] \textit{ is mutated by SBM}\right) &= \frac{1}{n}\\
E(\#\textit{bits mutated by SBM}) &= 1
\end{aligned}$$

Clearly, for all three variants the parameter r plays an important role. Clark et al. (2005) point out that avoiding the extreme cases $r = 0$ and $r = 1$ makes sense to avoid getting stuck. Yet, we decide to exclusively consider the extreme case $r = 1$, here. Note that for CHM$_2$ this rules out mutations not flipping any bits. Moreover, it rules out global convergence for all three variants. Consider for example the function $f\colon \{0,1\}^n \rightarrow \mathbb{R}$ ($n > 5$) with

$$f(x) = \begin{cases} \sum\limits_{i=0}^{n-1} x[i] & \text{if } x[0] = x[4],\\ -1 & \text{otherwise.} \end{cases}$$

The all one bit string 1^n is the unique global maximum. However, if $x[0] + x[4] = 0$ and $x[1] + x[2] + x[3] = 3$ both hold no somatic contiguous hypermutation with $r = 1$ can create this global optimum by means of a single mutation. It can only be reached via some other x' with smaller function value. If such moves are not accepted the algorithm is unable to optimize this simple function. We are aware of this consequence due to $r = 1$. In the following, we restrict our attention to functions where these limited capabilities of somatic contiguous hypermutations with $r = 1$ are not an issue.

Moreover, for many objective functions $f\colon \{0,1\}^n \to \mathbb{R}$ it is at some point of time essential that the search operator is able to change exactly some specific bits, say b many. We call such a mutation a specific *b-bit mutation*. In order to understand differences between somatic contiguous hypermutations and standard bit mutations it helps to see the difference in the probabilities for such specific b-bit mutations. Since we restricted our attention to the special case $r = 1$ we are only interested in specific b-bit mutations where the b bits form a contiguous region in the bit string x.

Standard bit mutations perform any such b-bit mutation with probability

$$\frac{1}{n^b} \cdot \left(1 - \frac{1}{n}\right)^{n-b} = \Theta\!\left(\frac{1}{n^b}\right)$$

since b specific bits need to flip, each with probability $1/n$, and the other $n - b$ bits must not flip, each with probability $1 - 1/n$. Since the bits are flipped independently the result follows. Note that this holds independently of the positions of the bits, also if they are not contiguous.

For somatic contiguous hypermutations things are entirely different. If the b bits that need to flip are not contiguous such a b-bit mutation cannot be performed in a single mutation and thus has probability 0 to occur. This is due to our extreme choice $r = 1$. If, on the other hand, the b bits are contiguous then the first and the last of these bits have to be chosen as the beginning and end of the contiguous region. For all three operators this is the case with probability $\Theta(1/n^2)$ for any $b \geq 1$ if the rightmost bit to be flipped is not the rightmost bit in x. In this special case the probability may be larger for CHM_1 (see Lemma 5.1 and 5.2) .

In the following we exclude functions where b-bit mutations are necessary that cannot occur with somatic contiguous hypermutations. The fact that these mutations are impossible results from our extreme choice of $r = 1$, any choice $0 < r < 1$ yields a positive probability for such mutations. It would be inappropriate and misleading to choose an extreme framework and then demonstrate drawbacks that are uniquely caused by this extreme choice. What we see for mutations that can be carried out is that their probability decreases exponentially with b for standard bit mutations while it is completely independent of b for somatic contiguous hypermutations. This leads to the speculation that for functions where 1-bit mutations are sufficient somatic contiguous hypermutations may be outperformed by standard bit mutations since such mutations occur with much smaller probability. On the other hand one may believe that for functions where b-bit mutations with $b > 2$ are necessary somatic contiguous hypermutations may excel and that the advantage may be tremendously large for $b \gg 2$. We investigate these speculations in the following.

5.2. General Bounds on the Optimization Time

Before considering concrete example functions, we derive a general lower bound on the expected optimization time for any objective function $f\colon \{0,1\}^n \to \mathbb{R}$ with a unique global optimum by means of the considerations in the previous section. We state this simple result here since it helps us to better understand the upper bounds that we derive in the following. Note that it is easy to generalize this bound to objective functions with more global optima. We do not present such a more general version since it does not add additional insight and complicates the proof.

Theorem 5.5 (Jansen and Zarges (2011a)). *Let $\mu = 1$ and $r = 1$. Moreover, let $f\colon \{0,1\}^n \to \mathbb{R}$ be given with a unique global optimum $x^* \in \{0,1\}^n$, i. e.,*

$$\left\{x \in \{0,1\}^n \mid f(x) = \max\left\{f(x') \mid x' \in \{0,1\}^n\right\}\right\} = \{x^*\}.$$

For any $x \in \{0,1\}^n$ with $x \neq x^*$ the expected optimization time of Algorithm 3.1 starting in x and using somatic contiguous hypermutations from Algorithm 5.1, 5.2, or 5.3, respectively, is

$$E(T_{CHM_1,f,x}) = \Omega(n) \tag{5.13}$$

$$E(T_{CHM_2,f,x}) = \Omega\bigl(n^2\bigr) \tag{5.14}$$

$$E(T_{CHM_3,f,x}) = \Omega\bigl(n^2\bigr) \tag{5.15}$$

Proof. According to our assumption we have $x \neq x^*$ and thus there is at least one mutation necessary. We consider the very last mutation in a run leading to x^*. For CHM$_2$ the values for $\min\{p_1,p_2\}$ and $\max\{p_1,p_2\}$ are uniquely defined. Thus, this mutation has probability at most $2/n^2$ and $\mathrm{E}\left(T_{\mathrm{CHM}_2,f,x}\right) \geq n^2/2 = \Omega\bigl(n^2\bigr)$ follows. For CHM$_3$ the values of p and l are uniquely defined. Thus, this mutation has probability $1/(n(n+1))$ and $\mathrm{E}\left(T_{\mathrm{CHM}_3,f,x}\right) \geq n^2 + n = \Omega\bigl(n^2\bigr)$ follows. For CHM$_1$ the value of p is also uniquely defined. For l, however, up to n values may be possible. Thus, this final mutation may have probability up to $(1/n) \cdot (n/(n+1))$ and $\mathrm{E}\left(T_{\mathrm{CHM}_1,f,x}\right) \geq n+1 = \Omega(n)$ follows. □

We observe that the lower bound for CHM$_1$ is by a factor of $\Theta(n)$ smaller than the bounds for the other two variants. This is due to the fact that CHM$_1$ truncates the contiguous region at the end of the bit string and thus allows for multiple values of l causing the same mutation if the contiguous region is at the end of the bit string. Since the proof of Theorem 5.5 is so simple only considering the very last mutation it is surprising to see that the lower bounds from Theorem 5.5 are all asymptotically tight in the following sense. There are a concrete function $f\colon \{0,1\}^n \to \mathbb{R}$ and $x_0 \in \{0,1\}^n$ such that the lower bounds from Theorem 5.5 match the corresponding upper bounds. We show this in the following theorem.

Theorem 5.6 (Jansen and Zarges (2011a)). *Let $\mu = 1$ and $r = 1$. Consider $f\colon \{0,1\}^n \to \mathbb{R}$ with*

$$f(x) = \begin{cases} 2 & \text{if } x = 1^n, \\ 1 & \text{if } x = 1^{n-1}0, \\ 0 & \text{otherwise} \end{cases}$$

and $x_0 = 1^{n-1}0$. *The expected optimization time of Algorithm 3.1 initialized in* x_0 *and using somatic contiguous hypermutations from Algorithm 5.1, 5.2, or 5.3, respectively, on* f *is*

$$\begin{aligned} E(T_{CHM_1,f,x_0}) &= n+1 \\ E(T_{CHM_2,f,x_0}) &= n^2 \\ E(T_{CHM_3,f,x_0}) &= n^2+n. \end{aligned}$$

Proof. The unique global optimum of f is $x^* = 1^n$, the all one bit string. The initial bit string $x_0 = 1^{n-1}0$ is the unique second best point in the search space, all other bit strings have worse function value. Thus, Algorithm 3.1 using somatic contiguous hypermutation stays in x_0 until a mutation to the unique global optimum 1^n is found. For CHM_1 such a mutation occurs for $p = n-1$ and $l > 0$. We see that it has probability $(1/n) \cdot n/(n+1)$ and $\text{E}(T_{\text{CHM}_1,f,x_0}) = n+1$ follows. For CHM_2 we need $p_1 = p_2 = n-1$ and see that $\text{E}(T_{\text{CHM}_2,f,x_0}) = n^2$ holds. For CHM_3 we need to have $p = n-1$ and $l = 1$, thus this mutation has probability $(1/n) \cdot 1/(n+1)$ and $\text{E}(T_{\text{CHM}_3,f,x_0}) = n^2+n$ follows. □

5.3. Results for OneMax

In the preceding chapter, we investigated the performance of different mutation operators on the example function OneMax. Thus it makes sense to start with an analysis of OneMax for somatic contiguous hypermutations. Recall, that the (1+1) EA using mutation probability $1/n$ requires optimization time $\Theta(n \log n)$ on OneMax (Droste et al. 2002). With somatic contiguous hypermutations it takes considerably longer to optimize this simple function.

The proof of the following theorem makes use of a simple drift theorem due to He and Yao (2004). In our context we have a distance measure $V\colon \{0,1\}^n \to \mathbb{R}_0^+$ such that $V(x) = 0$ holds if and only if x satisfies our optimization criterion. Given an upper bound on the expected decrease in distance c (the drift), the expected optimization time to reach the optimization goal is bounded below by $V(x_0)/c$ where $V(x_0)$ denotes the initial distance.

Theorem 5.7 (Jansen and Zarges (2011a))**.** *Let* $\mu = 1$ *and* $r = 1$. *The expected optimization time of Algorithm 3.1 using somatic contiguous hypermutations from Algorithm 5.1, 5.2, or 5.3, respectively, on* OneMax *is*

$$\begin{aligned} E(T_{CHM_1,\text{OneMax}}) &= O\big(n^2 \log n\big) \\ E(T_{CHM_2,\text{OneMax}}) &= \Theta\big(n^2 \log n\big) \\ E(T_{CHM_3,\text{OneMax}}) &= \Theta\big(n^2 \log n\big). \end{aligned}$$

Proof. The upper bound is easy to prove for all three kinds of somatic contiguous hypermutations using trivial fitness layers (Droste et al. 2002). For any $x \in \{0,1\}^n$ with $\text{OneMax}(x) = n - z$ there are exactly z 0-bits. If exactly one of these bits is flipped the function value is increased by exactly 1. Such a mutation occurs with probability $\Omega\bigl(1/n^2\bigr)$ for each of the z 0-bits and each mutation operator. Thus, the waiting time to increase the function value from $n-z$ to at least $n-z+1$ is bounded above by $O\bigl(n^2/z\bigr)$. Since function values cannot decrease due to the strict selection employed the expected optimization is bounded above by

$$\sum_{i=1}^{n} O\left(\frac{n^2}{z}\right) = O\left(n^2 \sum_{i=1}^{n} \frac{1}{z}\right) = O\bigl(n^2 \log n\bigr).$$

For the lower bound we only consider CHM_2 and CHM_3. We make use of drift arguments (He and Yao 2004) as described above. Before considering a drift measure V we consider an auxiliary measure V' in a first step. For $x \in \{0,1\}^n$ let $V'(x)$ denote the number of 0-bits in x. We bound the decrease in distance in a single mutation as given by V' from above. Let $x \in \{0,1\}^n$ denote the current bit string, let $x^+ \in \{0,1\}^n$ denote the bit string after one mutation and selection step. For CHM_2 and CHM_3, any specific b-bit mutation has probability $\Theta\bigl(1/n^2\bigr)$. To make an advance by i we need to mutate $i+a$ 0-bits from 0 to 1 and a 1-bits from 1 to 0 (for any $a \in \mathbb{N}_0$). If such a mutation is to have a positive probability the bits need to be arranged in an appropriate way. Initially, the probability for this decreases exponentially with i and a since the bits are initialized uniformly at random. Thus, the probability for l subsequent 0-bits equals $1/2^l$. Since we consider OneMax, the probability of being a 0-bit decreases with time. Thus,

$$\text{Prob}\left(V'(x) - V'(x^+) = i \mid x\right) = O\left(\frac{V'(x)}{n^2 \cdot k^i}\right)$$

holds for a constant $k \geq 2$ and $i > 0$. We conclude that we have

$$\begin{aligned}\text{E}\left(V'(x) - V'(x^+) \mid x\right) &= \sum_{i=1}^{n} i \cdot \text{Prob}\left(V'(x) - V'(x^+) = i \mid x\right)\\ &= O\left(\sum_{i=1}^{n} i \cdot \frac{V'(x)}{n^2 \cdot k^i}\right) = O\left(\frac{V'(x)}{n^2}\right)\end{aligned}$$

as upper bound on the drift measured by V'. Now we consider the situation from the point of view of the new search point. Note that we have $V'(x) > 0$ by assumption since otherwise optimization was already complete. However, we may have $V'(x^+) = 0$ (and actually do have this if the optimum is found in this step). Thus, we consider $V'(x^+) + 1$ instead of $V'(x^+)$ in order to avoid difficulties with the case $V'(x^+) = 0$. Since on expectation the Hamming distance between x and x^+ is bounded above by a positive constant less than 1 and since V' maps to $\mathbb{N}_0$, we have that also

$$\begin{aligned}\text{E}\left(V'(x) - V'(x^+) \mid x\right) &= O\left(\frac{V'(x)}{n^2}\right)\\ &= O\left(\frac{V'(x)}{\text{E}\left(V'(x^+) + 1 \mid x\right)} \cdot \frac{\text{E}\left(V'(x^+) + 1 \mid x\right)}{n^2}\right) = O\left(\frac{\text{E}\left(V'(x^+) + 1 \mid x\right)}{n^2}\right)\end{aligned}$$

holds.

For the application of the drift theorem (He and Yao 2004) we define another drift measure $V\colon \{0,1\}^n \to \mathbb{R}_0^+$ by $V(x) = H_{V'(x)}$ where $H_v = \sum_{i=1}^{v} 1/i$ denotes the v-th harmonic number (with $v \in \mathbb{N}_0$ and $H_0 = 0$, Lemma B.6).

We need to bound $\mathrm{E}\left(V(x) - V(x^+) \mid x\right) = \mathrm{E}\left(H_{V'(x)} - H_{V'(x^+)} \mid x\right)$ from above. Note that due to the strict plus-selection $V'(x) \geq V'(x^+)$. Consider $H_a - H_b$ for $a \geq b$. Clearly, $H_a - H_b = \sum_{i=b+1}^{a} 1/i$ holds and

$$\frac{a-b}{a} = \sum_{i=b+1}^{a} \frac{1}{a} \leq H_a - H_b \leq \sum_{i=b+1}^{a} \frac{1}{b+1} = \frac{a-b}{b+1}$$

follows. Using this and $\mathrm{E}\left(V'(x) - V'(x^+) \mid x\right) = O\big(\mathrm{E}\left(V'(x^+) + 1 \mid x\right)/n^2\big)$ from above we get

$$\begin{aligned}
&\mathrm{E}\left(V(x) - V(x^+) \mid x\right) = \mathrm{E}\left(H_{V'(x)} - H_{V'(x^+)} \mid x\right)\\
&\leq \mathrm{E}\left(\frac{V'(x) - V'(x^+)}{V'(x^+) + 1} \mid x\right)\\
&= O\left(\mathrm{E}\left(\frac{\mathrm{E}\left(V'(x^+) + 1 \mid x\right)}{n^2} \cdot \frac{1}{V'(x^+) + 1} \mid x\right)\right)\\
&= O\left(\frac{\mathrm{E}\left(V'(x^+) + 1 \mid x\right)}{n^2} \cdot \mathrm{E}\left(\frac{1}{V'(x^+) + 1} \mid x\right)\right)\\
&= O\left(\frac{\mathrm{E}\left(V'(x^+) + 1 \mid x\right)}{n^2} \cdot \sum_{i=0}^{V'(x)} \frac{\mathrm{Prob}\left(V'(x^+) = i \mid x\right)}{i+1}\right)\\
&= O\left(\frac{\mathrm{E}\left(V'(x^+) + 1 \mid x\right)}{n^2} \cdot \sum_{i=0}^{V'(x)} \frac{\mathrm{Prob}\left(V'(x) - V'(x^+) = V'(x) - i \mid x\right)}{i+1}\right)\\
&= O\left(\frac{\mathrm{E}\left(V'(x^+) + 1 \mid x\right)}{n^2 \cdot (V'(x) + 1)}\right)\\
&\quad + O\left(\sum_{i=0}^{V'(x)-1} \frac{\mathrm{E}\left(V'(x^+) + 1 \mid x\right)}{n^2} \cdot \frac{\mathrm{Prob}\left(V'(x) - V'(x^+) = V'(x) - i \mid x\right)}{i+1}\right)\\
&= O\left(\frac{1}{n^2}\right) + O\left(\sum_{i=0}^{V'(x)-1} \frac{\mathrm{E}\left(V'(x^+) + 1 \mid x\right)}{n^2} \cdot \frac{V'(x)}{n^2 \cdot k^{V'(x)-i} \cdot (i+1)}\right)\\
&= O\left(\frac{1}{n^2}\right) + O\left(\frac{1}{n^2} \cdot \sum_{i=0}^{V'(x)-1} \frac{1}{k^{V'(x)-i} \cdot (i+1)}\right)\\
&= O\left(\frac{1}{n^2}\right) + O\left(\frac{1}{n^2} \cdot \sum_{i=1}^{V'(x)} \frac{1}{k^i}\right) = O\left(\frac{1}{n^2}\right)
\end{aligned}$$

as upper bound on the drift. Since we have $0 \leq V'(x) \leq n$ we have $V(x) \leq H_n \leq \ln(n)+1$ as upper bound on the initial distance. Applying the drift theorem (He and Yao 2004) yields $\Omega\big(n^2 \log n\big)$ as lower bound on the expected optimization time. □

For CHM_1 we do not have probability $\Theta\big(1/n^2\big)$ for each b-bit mutation. Mutations at the end of the bit string can have much larger probabilities since for such mutations different values of length l lead to the same mutation. Note that this only applies to mutations that mutate the bits $x[p], x[p+1], \ldots, x[n-1]$. It is not easy to see how such mutations can lead to a drastic decrease in expected optimization time. We thus speculate that the upper bound $O\big(n^2 \log n\big)$ may be asymptotically tight for CHM_1, too. This speculation is supported by the results of experiments described later in Section 5.8.1.

For CHM_2 and CHM_3 with $r = 1$ we know that we lose a factor of $\Theta(n)$ in comparison to standard bit mutations (Theorem 5.7). The proof of the lower bound, however, relies on the fact that the initial bit string is chosen uniformly at random. One may wonder what happens if the initial bit string happened to be the all-zero bit string 0^n. With only 0-bits in the initial bit string there is a much bigger chance for larger increases in the number of 1-bits. However, this advantage is reduced over time as the 1-bits will be distributed randomly. It is unclear if this advantage that is big in the beginning where it is easy to make progress and decreased in the end where making progress becomes much harder anyway is sufficient to yield an asymptotically smaller expected optimization time. To get an impression we provide results of experiments in Section 5.8.1.

5.4. Results for LeadingOnes

The result on OneMax may lead to the belief that somatic contiguous hypermutations increase the expected optimization time by a factor of $\Theta(n)$ for objective functions where mutations of single bits are responsible for optimization. We demonstrate that things are not so simple by considering another well-known example function, namely LeadingOnes that yields as function value the number of consecutive 1-bits counted from left to right. It can formally be defined as follows.

Definition 5.8 (Rudolph (1997))**.** *For $n \in \mathbb{N}$ and $x \in \{0,1\}^n$, the function* LeadingOnes: $\{0,1\}^n \to \mathbb{R}$ *is defined by*

$$\text{LeadingOnes}(x) = \sum_{i=1}^{n} \prod_{j=1}^{i} x[j].$$

As for OneMax, the unique global optimum is the all one bit string 1^n. The expected optimization time when using standard bit mutations equals $\mathrm{E}\left(T_{\text{SBM},\text{LeadingOnes}}\right) = \Theta\big(n^2\big)$ (Droste et al. 2002) and this can be achieved with 1-bit mutations, only. While the expected optimization time can be larger when using somatic contiguous hypermutations it is much smaller than $\Theta\big(n^3\big)$.

Theorem 5.9 (Jansen and Zarges (2011a))**.** *Let $\mu = 1$ and $r = 1$. The expected optimization time of Algorithm 3.1 using somatic contiguous hypermutations from Algorithm 5.1, 5.2, or 5.3, respectively, on* LeadingOnes *is*

$$\begin{aligned} E(T_{CHM_1,\text{LeadingOnes}}) &= O(n^2) \\ E(T_{CHM_2,\text{LeadingOnes}}) &= \Theta(n^2 \log n) \\ E(T_{CHM_3,\text{LeadingOnes}}) &= \Theta(n^2 \log n). \end{aligned}$$

Proof. Again, we prove the upper bound using trivial fitness layers (Droste et al. 2002). For any $x \in \{0,1\}^n \setminus \{1^n\}$ it suffices to mutate the leftmost 0-bit and an arbitrary number of bits to its right. Let the position of the leftmost 0-bit be $n-i-1$, thus there are i bits to its right.

For CHM$_1$ it suffices to have $p = n-i-1$ and $l > 0$, thus the probability for an improving mutation is bounded below by $(1/n) \cdot (1 - 1/(n+1)) = \Omega(1/n)$. This yields $O(n \cdot n) = O(n^2)$ as upper bound on the expected optimization time.

For CHM$_2$ we need $\min\{p_1, p_2\} = n-i-1$ and have $i+1$ positions for $\max\{p_1, p_2\}$. This happens with probability $\Omega((i+1)/n^2)$ and thus yields $O(n^2/(i+1))$ as upper bound for the expected waiting time for such a mutation. Thus, the expected optimization is bounded above by

$$\sum_{i=0}^{n-1} O\left(\frac{n^2}{i+1}\right) = O\left(n^2 \sum_{i=1}^{n} \frac{1}{i}\right) = O(n^2 \log n).$$

For CHM$_3$ we need $p = n-i-1$ and $0 < l \leq i$. Thus we have $i/(n^2+n)$ as probability for an improving mutation and we obtain $\sum\limits_{i=1}^{n} O(n^2/i) = O(n^2 \log n)$ as upper bound on the expected optimization time.

For the lower bound for CHM$_2$ and CHM$_3$ we can observe that the bits to the right of the leftmost 0-bit are distributed uniformly at random. Thus, the probability to increase the function value by j is bounded above by $O((i+1)/(n^2 2^{j-1}))$. Again making use of drift arguments (He and Yao 2004) we see that we obtain a lower bound of $\Omega(n^2 \log n)$ on the expected optimization time. □

As for OneMax, for CHM$_1$ we do not have a matching lower bound. This is also due to the much higher probabilities for mutations that mutate the bits $x[p], x[p+1], \ldots, x[n-1]$. We again speculate that the upper bound $O(n^2)$ may be asymptotically tight for CHM$_1$, too. This, again, is supported by experiments (see Section 5.8.2).

For LeadingOnes, the use of somatic contiguous hypermutations implies a decrease in performance by a factor of $O(\log n)$ in comparison to standard bit mutations. As for OneMax, one may wonder what happens if the initial bit string happened to be the all-zero bit string 0^n. While this constitutes an advantage in the beginning we observe that any mutation not affecting the first $i+1$ bits in a current bit string x with LeadingOnes$(x) = i$ will be accepted. This leads to a random distribution of the bits that are right of the leftmost 0-bit and decreases the initial advantage. As for OneMax, we investigate both questions by doing experiments in Section 5.8.2.

5.5. Results for $n \cdot$ LeadingOnes – OneMax

In the preceding sections on OneMax and LeadingOnes, we speculated that it might help to initialize deterministically in 0^n. However, we did not prove upper and lower bounds for this case and referred to experiments in Section 5.8. For LeadingOnes one observation was that the bits that are right to the leftmost 0-bit become randomly distributed. In particular, the result of the very first mutation is accepted in any case and thus, we leave 0^n immediately. In the following, we consider a modification of LeadingOnes where this is not the case. The function LeadingOnes′ can formally be defined as follows.

Definition 5.10. *For $n \in \mathbb{N}$ and $x \in \{0,1\}^n$, the function* LeadingOnes′: $\{0,1\}^n \to \mathbb{R}$ *is defined by*

$$\text{LeadingOnes}'(x) = n \cdot \text{LeadingOnes}(x) - \text{OneMax}(x).$$

Since the selection mechanism that we employ is insensitive to the absolute function values and reacts to the ordering of function values only, we do not change anything by going from LeadingOnes to $n \cdot$ LeadingOnes. Going from $n \cdot$ LeadingOnes to LeadingOnes′ $= n \cdot$ LeadingOnes – OneMax has the following effects. First, since the number of leading 1-bits comes with the factor n it is the number of leading 1-bits that dominates the function value. A bit string with a larger number of leading 1-bits has the larger function value. For bit strings with equal numbers of leading 1-bits, it is the number of 1-bits that decides. Fewer 1-bits imply larger function values. Thus, for CHM$_1$ and CHM$_2$ with $r = 1$, starting with 1^i0^{n-i}, the current bit string will always be of the form $x = 1^j0^{n-j}$ ($j \geq i$) with LeadingOnes$(x) = j$ and LeadingOnes′(x) $= n \cdot j - j = (n-1) \cdot j$. This suffices to demonstrate a noticeable advantage for most somatic contiguous hypermutations when started in 0^n.

Theorem 5.11 (Jansen and Zarges (2011a)). *Let $\mu = 1$ and $r = 1$. The expected optimization time of Algorithm 3.1 using somatic contiguous hypermutations from Algorithm 5.1, 5.2, or 5.3, respectively, on* LeadingOnes′ *is*

$$\begin{aligned} &E\left(T_{CHM_1,\text{LeadingOnes}',0^n}\right) = O(n) \\ n^2/2 \ \leq &E\left(T_{CHM_2,\text{LeadingOnes}',0^n}\right) \leq n^2 \\ &E\left(T_{CHM_3,\text{LeadingOnes}',0^n}\right) = O\left(n^2 \log n\right). \end{aligned}$$

Proof. For CHM$_1$ and CHM$_2$ we have at any time $x = 1^i0^{n-i}$ for some $i \in \{0,1,\ldots,n\}$ where x is the current bit string. A mutation can only change this if none of the i leading 1-bits is mutated, the 0-bit at position $x[i]$ is mutated and any number of the $n-i-1$ consecutive 0-bits are mutated. For CHM$_3$ this is not true since a mutation of 0^n may lead to $1^i0^j1^{n-i-j}$.

For CHM$_1$ we observe that we have $p = i$ with probability $1/n$ and $l \geq n/3$ with probability at least $\lfloor (2/3)n \rfloor/(n+1) > 1/2$. Thus, with probability at least $1/(2n)$ the number of trailing 0-bits is reduced either by at least $n/3$ or to 0. Clearly, the expected waiting time for such a mutation is bounded above by $2n$ and after at most three such mutations the number of 0-bits is reduced to 0. This implies $\mathrm{E}\left(T_{\mathrm{CHM}_1,\textsc{LeadingOnes}'}\right) = O(n)$.

For CHM_2 we observe that for $x \neq 1^{n-1}0$ we can either have $p_1 = i$ and $p_2 = n-1$ or $p_1 = n-1$ and $p_2 = i$ for a mutation that reaches the optimum. Such a mutation has thus probability $2/n^2$ in this situation. Since no other mutations can lead to the global optimum this implies that the expected optimization time is bounded below by $n^2/2$. For $x = 1^{n-1}0$ the only mutation leading to the unique global optimum 1^n has $p_1 = p_2 = n-1$. This mutation has probability $1/n^2$ and $\mathrm{E}\left(T_{\mathrm{CHM}_2,\textsc{LeadingOnes}}\right) \leq n^2$ follows.

For CHM_3 we observe that for the current string there is exactly one value of p such that a mutation increases the number of leading 1-bits. Moreover, the mutation increases the function value if and only if $0 < l \leq n - i$ holds if i denotes the number of leading 1-bits. Thus, if the number of leading 1-bits equals i it is increased by at least 1 with probability $(1/n) \cdot (n-i)/(n+1)$. Thus, the expected optimization time is bounded above by

$$\sum_{i=0}^{n-1} \frac{n \cdot (n+1)}{n-i} = (n^2 + n) \sum_{i=1}^{n} \frac{1}{i} = O\big(n^2 \log n\big). \qquad \square$$

Like before we present empirical results in Section 5.8.3.

5.6. Results for $n \cdot$ TrailingOnes – OneMax

The most noticeable result on LeadingOnes′ is the tremendous advantage for CHM_1. We already pointed out that CHM_1 has a very strong positional bias, a much stronger bias than CHM_2 and CHM_3, the latter being completely unbiased with respect to positions of bits. Clearly, such bias is undesirable as long as one has no reason to assume that this bias matches properties of the objective function (Droste and Wiesmann 2003). Yet here it leads to a significant advantage. It is easy to see why that is the case. The variant CHM_1 has a strong tendency to mutate bits further to the right. In LeadingOnes′ we have that the bits to the left tend to be correct early in the run and it pays off to concentrate on flipping the bits on the right hand side. We demonstrate that the advantage for CHM_1 is due to this property of LeadingOnes′ by considering another example function that is almost identical with LeadingOnes′ but with a reversed bit string. We consider the function TrailingOnes’ defined formally in the following.

Definition 5.12. *For $n \in \mathbb{N}$ and $x \in \{0,1\}^n$, the function* TrailingOnes′: $\{0,1\}^n \to \mathbb{R}$ *is defined by*

$$\textsc{TrailingOnes}'(x) = n \cdot \left(\sum_{i=1}^{n} \prod_{j=n-i}^{n-1} x[j] \right) - \textsc{OneMax}(x).$$

While the function value of $\textsc{LeadingOnes}'$ is dominated by the number of leading 1-bits in $\textsc{TrailingOnes}'$ it is dominated by the number of trailing 1-bits. For $x = (x[0], x[1], \ldots, x[n-1]) \in \{0,1\}^n$ let $x^R = (x[n-1], x[n-2], \ldots, x[0])$. Clearly, $\textsc{LeadingOnes}'(x) = \textsc{TrailingOnes}'(x^R)$ holds for all $x \in \{0,1\}^n$. Since CHM_2 and CHM_3 are completely symmetric for $x[i]$ and $x[n-i-1]$ we get as immediate consequence

$$\begin{aligned} \mathrm{E}\left(T_{\mathrm{CHM}_2,\textsc{TrailingOnes}'}\right) &= \mathrm{E}\left(T_{\mathrm{CHM}_2,\textsc{LeadingOnes}'}\right) \qquad \text{and} \\ \mathrm{E}\left(T_{\mathrm{CHM}_3,\textsc{TrailingOnes}'}\right) &= \mathrm{E}\left(T_{\mathrm{CHM}_3,\textsc{LeadingOnes}'}\right). \end{aligned}$$

For CHM_1 this is not the case. While Theorem 5.11 yields $\mathrm{E}\left(T_{\mathrm{CHM}_1,\textsc{LeadingOnes}',0^n}\right) = O(n)$, CHM_1 is clearly slower on $\textsc{TrailingOnes}'$ as the following theorem shows.

Theorem 5.13. *Let $\mu = 1$ and $r = 1$. The expected optimization time of Algorithm 3.1 using somatic contiguous hypermutations from Algorithm 5.1 on* $\textsc{TrailingOnes}$*' is*

$$E\left(T_{CHM_1,\textsc{TrailingOnes}',0^n}\right) \geq n^2 - 1.$$

Proof. The first mutation changing the current bit string affects the rightmost bit $x[n-1]$, i. e., in this mutation $p + l - 1 \geq n - 1$ holds for the random position p and the random length l of the interval to be mutated. With probability $(n-1)/n$ we have $p > 0$ in this mutation. After this mutation the current bit string is $0^i 1^{n-i}$ for some $i \in \{1, 2, \ldots, n-1\}$. Due to the definition of $\textsc{TrailingOnes}'$ we have $x = 0^{i'} 1^{n-i'}$ with $0 \leq i' \leq i$ for all subsequent current bit strings x. This implies that there is always exactly one mutation that leads to the global optimum 1^n. For each of these uniquely defined mutations the probability equals $(1/n) \cdot 1/(n+1)$. Thus we have

$$\mathrm{E}\left(T_{\mathrm{CHM}_1,\textsc{TrailingOnes}',0^n}\right) \geq \frac{n-1}{n} \cdot (n \cdot (n+1)) = n^2 - 1. \qquad \square$$

We present the results of experiments for $\textsc{TrailingOnes}'$ as we did for the other functions in Section 5.8.4.

5.7. Results for $\mathrm{CLOB}_{b,k}$

When discussing probabilities of specific b-bit mutations we observed that somatic contiguous hypermutations can be advantageous if feasible b-bit mutations for $b > 2$ are needed and that this advantage grows exponentially with b. In the following, we consider a function designed for this purpose in a different context and for a different algorithm.

We consider a function similar to $\textsc{LeadingOnes}$ but replacing the role of single leading 1-bits by blocks of leading 1-bits of equal length b. Thus, for $b = 1$ we have $\textsc{LeadingOnes}$ where single bit mutations are sufficient, for $b > 1$ we have a function where the function value can be increased by a mutation of b bits. Such an example function could be called $\textsc{LeadingOnesBlocks}_b$ or LOB_b, for short. Since we want to simplify the analysis we would like to make sure that exactly b-bit mutations are needed. This can be achieved by moving to $n \cdot \mathrm{LOB}_b - \textsc{OneMax}$ just as we moved from $\textsc{LeadingOnes}$ to $\textsc{LeadingOnes}'$.

As we mentioned above, such a function was introduced in a different context. Since we want to avoid 'inventing' new example functions (where no results for comparisons are known) we stick to the definition of this known example function even though it is a little bit more involved than what is actually needed here. The function is called $\text{CLOB}_{b,k}$ (short for $\text{ConcatenatedLeadingOnesBlocks}_{b,k}$) and is defined as k independent copies of $n \cdot \text{LOB}_b$ where the complete function value is given by adding up the function values of the k copies. The function was originally introduced by Jansen and Wiegand (2004a) for the analysis of a simple cooperative coevolutionary algorithm, called the CC (1+1) EA. We proceed with a formal definition of this fitness function and a discussion of its main properties.

Definition 5.14. *For $b, k, n \in \mathbb{N}$ with $n/k \in \mathbb{N}$ and $n/(bk) \in \mathbb{N}$ and $x \in \{0,1\}^n$, the function $\text{CLOB}_{b,k}\colon \{0,1\}^n \to \mathbb{R}$ is defined by*

$$\text{CLOB}_{b,k}(x) = n \cdot \left(\sum_{h=1}^{k} \sum_{i=1}^{\frac{n}{bk}} \prod_{j=0}^{i \cdot b-1} x\left[(h-1) \cdot \frac{n}{k} + j\right] \right) - \text{OneMax}(x).$$

Obviously, it is defined as sum of k independent copies of the same function, operating on consecutive disjoint pieces of the bit string x, each of length n/k. To simplify notation a bit in the following we define $l := n/k$. Note that we have $l \in \mathbb{N}$. The function has the all one string 1^n as its unique global optimum.

Think of $x = (x[0], x[1], \dots, x[n-1]) \in \{0,1\}^n$ as divided into k pieces $x^{(1)}$, $x^{(2)}$, …, $x^{(k)}$ with $x^{(i)} = (x\left[(i-1) \cdot l\right], x\left[(i-1) \cdot l + 1\right], \dots, x\left[i \cdot l - 1\right]) \in \{0,1\}^l$ for each $i \in \{1, 2, \dots, k\}$. Each piece $x^{(i)}$ can be thought of as being divided into l/b consecutive disjoint blocks of length b each. In each piece the number of these blocks completely set to 1 is counted from left to right stopping with the first block different from 1^b. For each of these leading 1-blocks the function value is increased by n. We add this up for all k pieces and subtract the number of 1-bits.

Increasing the number of pieces k while keeping the length of the blocks b fixed decreases the length of each piece and decreases the number of blocks in each piece. Increasing b while keeping k fixed decreases the number of blocks in each piece without changing the length of the pieces.

It is known (Jansen and Wiegand 2004a) that the expected optimization time of the (1+1) EA on $\text{CLOB}_{b,k}$ equals

$$\text{E}\left(T_{(1+1)\text{ EA},\text{CLOB}_{b,k}}\right) = \Theta\left(n^b \left(\frac{l}{b} + \ln k\right)\right).$$

The cooperative coevolutionary algorithm analyzed by Jansen and Wiegand (2004a), the CC (1+1) EA, achieves an expected optimization time of

$$\text{E}\left(T_{\text{CC }(1+1)\text{ EA},\text{CLOB}_{b,k}}\right) = \Theta\left(k \cdot l^b \left(\frac{l}{b} + \ln k\right)\right)$$

beating the (1+1) EA by a factor of $\Theta\left(k^{b-1}\right)$. Note that this algorithm is provided with the information about the number k of pieces and their length l. This is different for the algorithm using somatic contiguous hypermutations. However, we are able to prove a bound that is even independent of k, b, and l.

Theorem 5.15 (Jansen and Zarges (2011a)). *Let $\mu = 1$ and $r = 1$. Moreover, let for $n \in \mathbb{N}$ the parameters $b, k \in \mathbb{N}$ be given with $n/(bk) \in \mathbb{N}$. The expected optimization time of Algorithm 3.1 using somatic contiguous hypermutations from Algorithm 5.1, 5.2, or 5.3, respectively, on $\text{CLOB}_{b,k}$ is*

$$\begin{aligned} E\left(T_{CHM_1,\text{CLOB}_{b,k}}\right) &= O\big(n^2 \log n\big) \\ E\left(T_{CHM_2,\text{CLOB}_{b,k}}\right) &= O\big(n^2 \log n\big) \\ E\left(T_{CHM_3,\text{CLOB}_{b,k}}\right) &= O\big(n^2 \log n\big). \end{aligned}$$

Proof. We remember the k pieces $x^{(1)}, x^{(2)}, \ldots, x^{(k)}$ of length $l := n/k$ each that together form $x \in \{0,1\}^n$. We know that on average after $O\big(n^2 \log n\big)$ steps the optimum of OneMax is reached. We conclude that on average for each of the k pieces after that many steps a bit string of the form $1^{i \cdot b} 0^{l - i \cdot b}$ (with possibly different values i for the different pieces) is reached. Then we are in a situation very similar to LeadingOnes′ $= n \cdot$LeadingOnes$-$OneMax. For each of the j pieces that are different from 1^l there is always a mutation creating 1^l. Each of these mutations occurs with probability $\Omega\big(1/n^2\big)$ for all three variants of somatic contiguous hypermutations. We thus obtain $\sum\limits_{j=1}^{k} n^2/j$ $= O\big(n^2 \log k\big)$ as upper bound. Since $k \leq n$ holds, this proves the upper bound. ☐

We observe that when we initialize deterministically in 0^n we are in this special situation having $1^{i \cdot b} 0^{l - i \cdot b}$ in each of the k pieces right at the start. Thus, the first phase where we wait for this to happen is empty. This reduces the upper bound that we are able to prove from $O\big(n^2 \log n\big)$ to $O\big(n^2 \log k\big)$. Again, we present empirical results for this function in Section 5.8.5.

5.8. Experimental Supplements

As announced in the previous sections, we present results of experiments in order to complement our theoretical analysis. For each function considered before, we perform 100 independent runs of the respective algorithm using random initialization as well as deterministic initialization in 0^n for for $n \in \{40, 80, 120, \ldots, 600\}$. We again plot the results using box-and-whisker plots (Definition B.1) In addition we plot $c \cdot b(n)$ if we have a bound $\mathrm{E}\left(T_{\text{op},f}\right) = O(b(n))$ for illustrative purposes (for some mutation operator op and some objective function f). The constant c is determined using a least squares fit. For the sake of completeness we plot analogous data for the algorithm using standard bit mutations if reasonable. In order to gain more insight we also plot the quotients $\mathrm{E}\left(T_{\text{op},f}\right) / \mathrm{E}\left(T_{\text{op},f,0^n}\right)$ (of the observed medians, of course) for all mutation operators op and functions f considered.

5.8.1. OneMax

The results for OneMax and the different mutation operators can be found in Figure 5.2. Note that the different scaling for standard bit mutations is due to the much smaller expected optimization time.

We observe that all variants of somatic contiguous hypermutations benefit from initialization in 0^n whereas for standard bit mutations there is hardly any difference. If this difference is so large as to constitute an asymptotic difference in the expected optimization time is impossible to tell from this graph, of course. Since in Figure 5.2 this quotient is bounded by two and does not appear to grow with n we speculate that starting with the all-zero bit string 0^n does not reduce the order of the expected optimization time. Since we observe no noticeable advantage for CHM_1 we speculate that also for $\mathrm{E}\left(T_{\text{CHM}_1,\text{OneMax}}\right)$ a lower bound of order $\Omega\big(n^2 \log n\big)$ can be proven.

5.8.2. LeadingOnes

Results for LeadingOnes are depicted in Figure 5.3. Note again, the different scaling in the box-and-whisker plots for standard bit mutations and CHM_1.

We observe for LeadingOnes that both, somatic contiguous hypermutations as well as standard bit mutations, are not sensitive at all with respect to the initialization method. In particular, in contradiction to our intuition, somatic contiguous hypermutations do not benefit from deterministic initialization in 0^n in a visible way. The difference is even smaller than for OneMax as the quotients clearly indicate. Obviously, the bits right to the leftmost 0-bit become randomly distributed so fast in comparison with the expected optimization time that no noticeable advantage is gained. This is different to the situation for OneMax. One difference between those two example functions is that the relevant bits for increasing the function value are gathered right to the leftmost 0-bit for LeadingOnes while they are randomly distributed among all bits for OneMax. This makes them more difficult to change for somatic contiguous hypermutations so that the initial advantage due to deterministic initialization in 0^n is longer preserved for OneMax leading to visible effects.

We observe that the average optimization time with CHM_1 is smaller than with SBM. It is known that the expected optimization time using SBM is bounded below by $0.859n^2$ (Böttcher et al. 2010). Remember, that for CHM_1 the probability of increasing the number of leading 1-bits equals $1/(n+1)$. Due to the random initialization we expect to be needing $n/2$ such improvements. This yields an expected optimization time of $0.5n^2 + 0.5n < 0.859n^2$.

5.8.3. $n \cdot$ LeadingOnes − OneMax

When considering LeadingOnes it is in some sense disappointing that initializing deterministically in 0^n helps so little. This becomes different for LeadingOnes if the bits right of the leftmost 0-bit do not become randomly distributed, i. e., for LeadingOnes′. We also investigate this function experimentally and present the results in Figure 5.4.

Considering the quotients we see a very clear benefit for CHM_1 due to initializing in the all-zero bit string 0^n. For CHM_3 no such effect is visible. For CHM_2, things are less clear. We additionally display the quotients $\mathrm{E}\left(T_{\mathrm{CHM}_i,\textsc{LeadingOnes}'}\right)/\mathrm{E}\left(T_{\mathrm{CHM}_i,\textsc{LeadingOnes}',0^n}\right)$ for $i \in \{2,3\}$. There may be a growing benefit for CHM_2 due to deterministic initialization in the all-zero bit string 0^n. We observe that the quotient grows from approximately 2.5 for $n = 40$ to approximately 4 for $n = 600$. If there really is an advantage of order $\omega(1)$ can only be decided by a theoretical analysis and is subject of future research.

Remember that, we have $n^2/2 \leq \mathrm{E}\left(T_{\mathrm{CHM}_2,\textsc{LeadingOnes}',0^n}\right) \leq n^2$ (Theorem 5.11). Even more, we showed that the expected optimization time is $n^2/2$ if the bit string $x = 1^{n-1}0$ is not reached, and n^2 otherwise. Thus, the expected optimization time is much closer to $n^2/2$ than to n^2. We see that the observed median in the experiments is $0.44n^2$. One might think that this contradicts our theoretical result. However, we also see a large number of large outliers in the experiments. Thus, we conclude that this observation might be inaccurate and that a larger number of runs will bring the observed median closer to $n^2/2$.

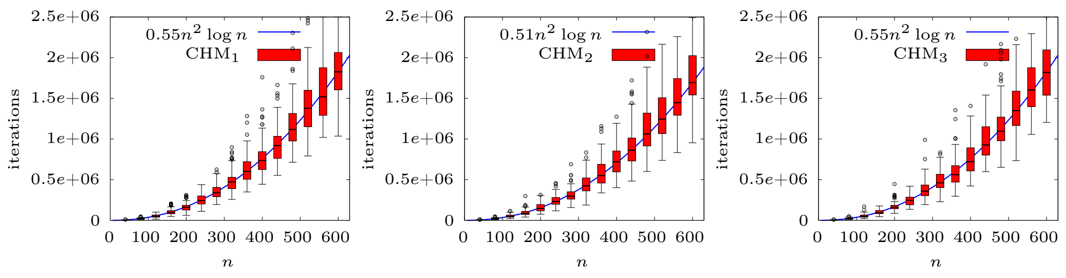
5.8.4. $n \cdot$ TrailingOnes – OneMax

We present the results of experiments for $\textsc{TrailingOnes}'$ in Figure 5.5. Here the comparison with the results for $\textsc{LeadingOnes}'$ (Figure 5.4) is most revealing. We see that while CHM_2, CHM_3, and standard bit mutations perform equally on $\textsc{LeadingOnes}'$ and $\textsc{TrailingOnes}'$ the advantage that CHM_1 has on $\textsc{LeadingOnes}'$ is not present on $\textsc{TrailingOnes}'$. This underlines the statement that a bias in a mutation operator is only desirable if it is well aligned with the objective function's properties.

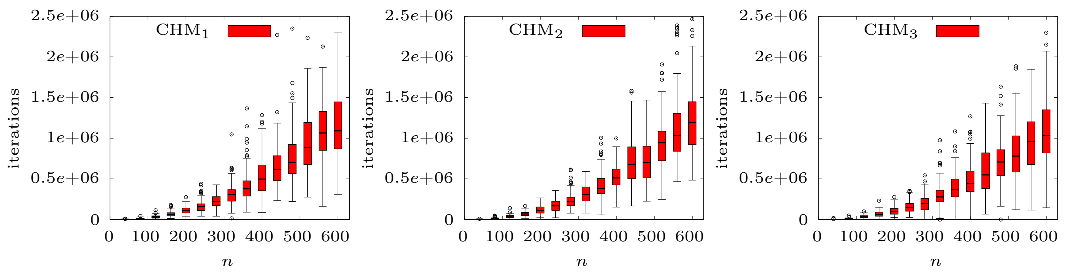
5.8.5. $\mathrm{CLOB}_{b,k}$

Finally, we present results for $\mathrm{CLOB}_{b,k}$ in Figure 5.6 and 5.7 for $b = 4$ and in Figure 5.8 and 5.9 for $b = 8$. We choose (quite arbitrarily) $k \in \{5, 10\}$ and $b \in \{4, 8\}$, so that we consider in total four version of $\mathrm{CLOB}_{b,k}$, namely $\mathrm{CLOB}_{4,5}$, $\mathrm{CLOB}_{8,5}$, $\mathrm{CLOB}_{4,10}$, and $\mathrm{CLOB}_{8,10}$. Note that for $\mathrm{CLOB}_{8,10}$ we need $n/80 \in \mathbb{N}$ to hold so that we only present results for $n \in \{80, 160, \ldots, 560\}$. Since the expected optimization times when using standard bit mutations are extremely large ($\Omega\big(n^4\big)$ for $b = 4$ and $\Omega\big(n^8\big)$ for $b = 8$) we do not present actual results for the (1+1) EA here. We remark that when performing such runs using standard bit mutations and considering for example $\mathrm{CLOB}_{4,5}$ the minimum observed optimization time in 100 runs for $n = 40$ when using standard bit mutations (3,896,445) is significantly larger than the maximum observed optimization time in 100 runs for $n = 600$ when using any of the three variants of somatic contiguous hypermutations (3,280,958). The maximum observed optimization time in 100 runs on $\mathrm{CLOB}_{4,5}$ using any of the three variants of somatic contiguous hypermutations for the same value $n = 40$ even equals only 8,328. The comparison is pointless, performing the runs for the (1+1) EA for these values of b and k not even feasible.

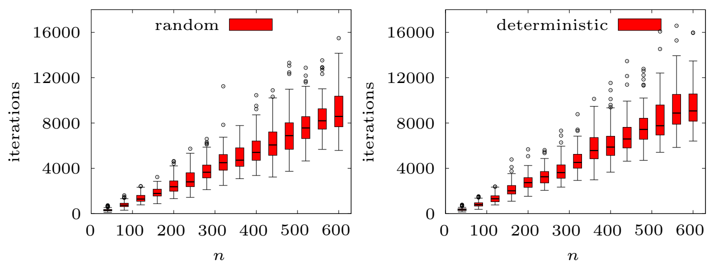
We observe that the median run times increase with b. This seems to contradict our upper bounds that are independent of b. Remember that we only have upper bounds and in our proofs crudely estimated the time spent in the $n-$ OneMax-phase by $O\left(n^2 \log n\right)$. However, in this phase, for smaller values of b the probability to already set rapidly leading one blocks is larger. This explains the faster optimization.



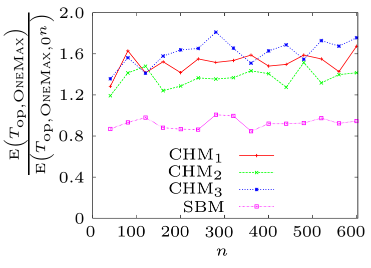
(a) random initialization



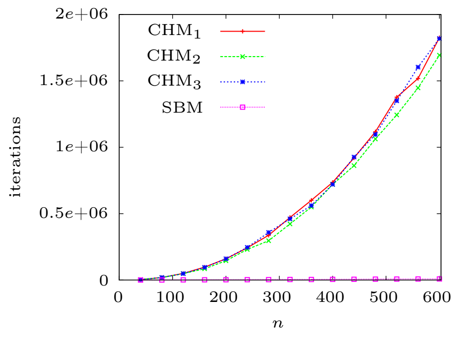
(b) deterministic initialization



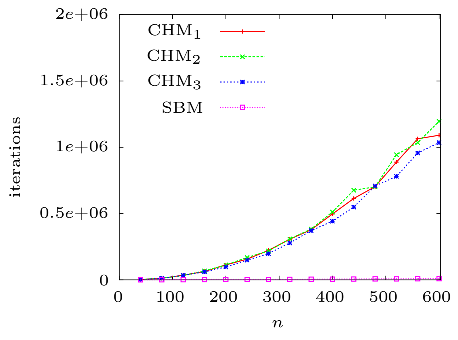
(c) standard bit mutations



(d) observed quotients

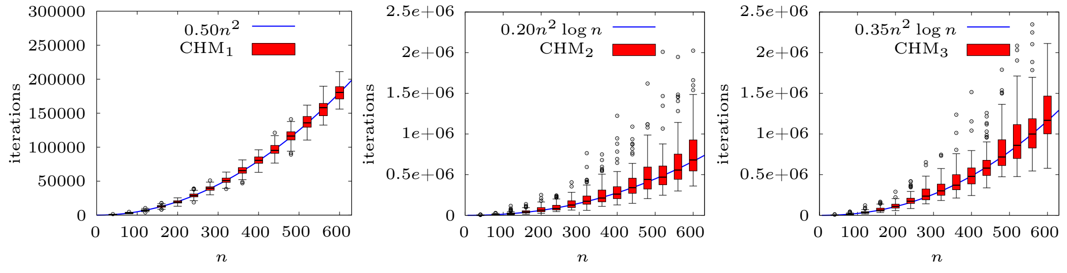


(e) medians, random

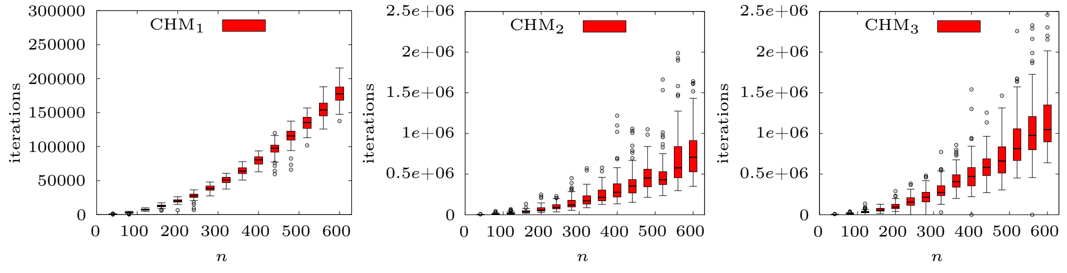


(f) medians, deterministic

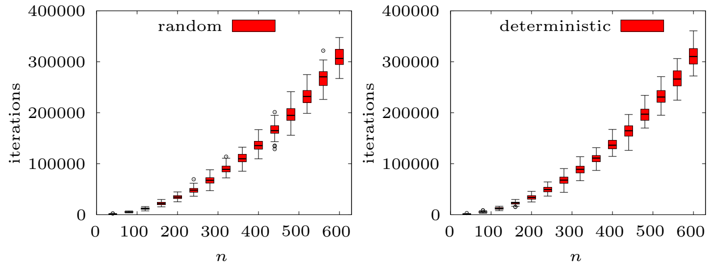
Figure 5.2.: Empirical data from 100 runs for OneMax.



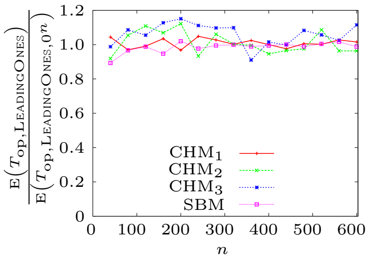
(a) random initialization



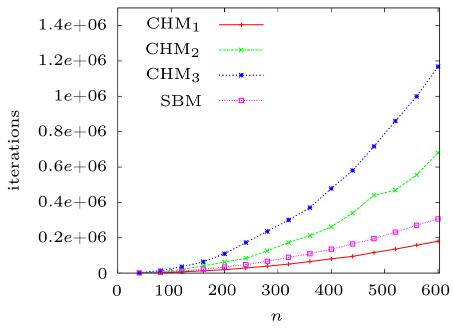
(b) deterministic initialization



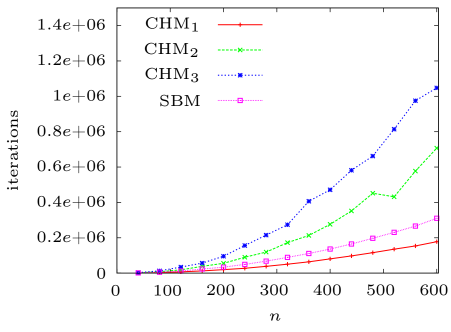
(c) standard bit mutations



(d) observed quotients

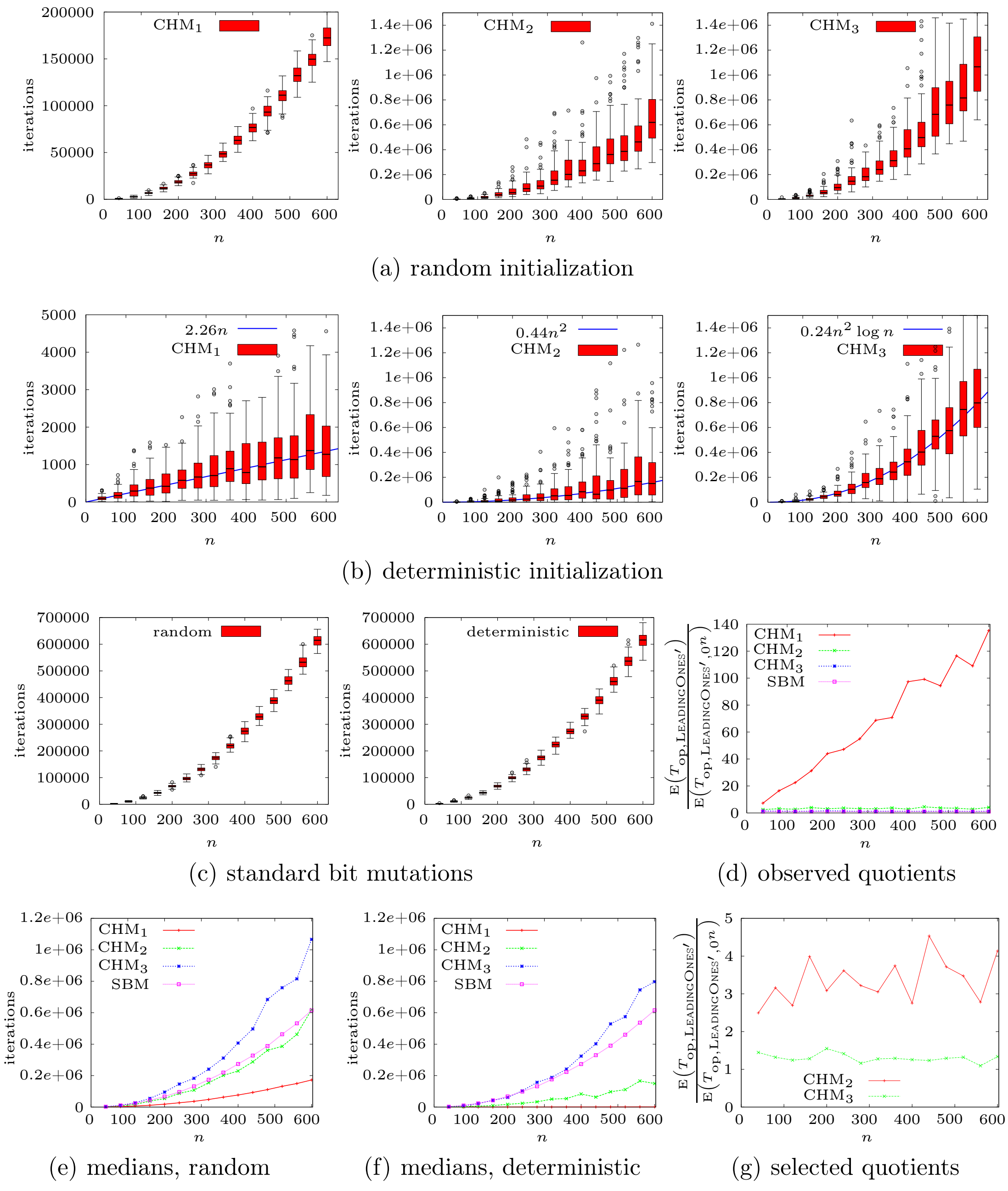


(e) medians, random



(f) medians, deterministic

Figure 5.3.: Empirical data from 100 runs for LeadingOnes.



(a) random initialization

(b) deterministic initialization

(c) standard bit mutations

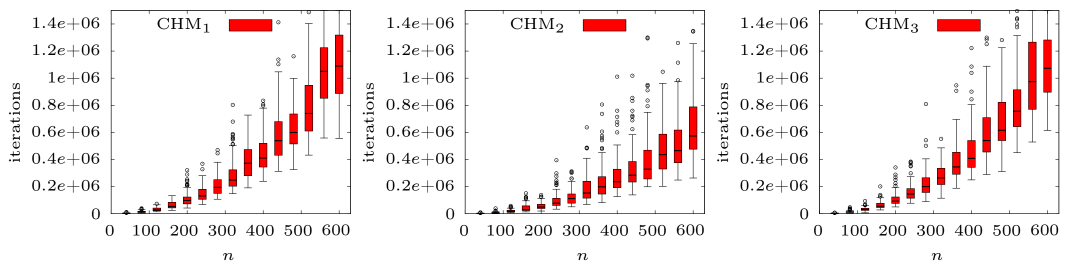
(d) observed quotients

(e) medians, random

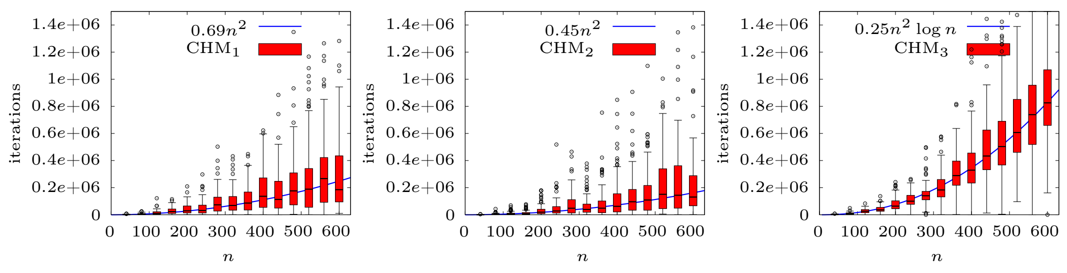
(f) medians, deterministic

(g) selected quotients

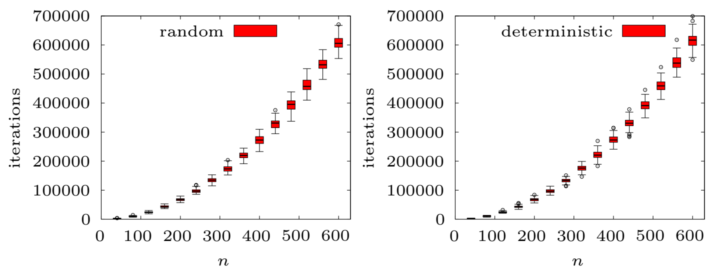
Figure 5.4.: Empirical data from 100 runs for LeadingOnes′.



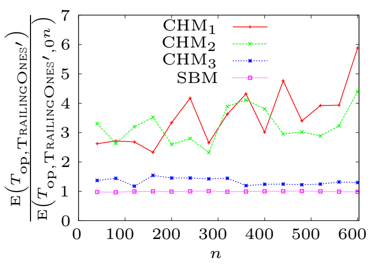
(a) random initialization



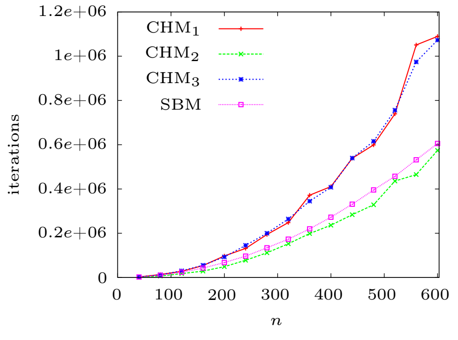
(b) deterministic initialization



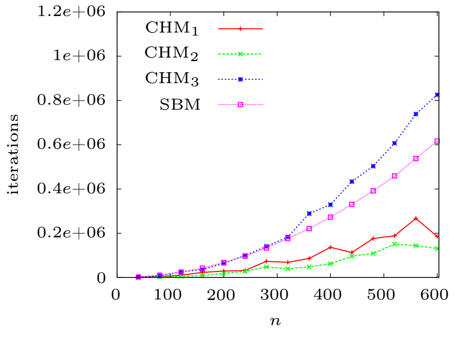
(c) standard bit mutations



(d) observed quotients

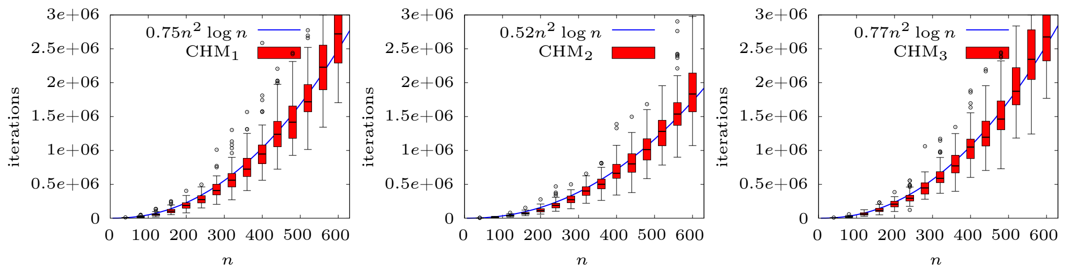


(e) medians, random

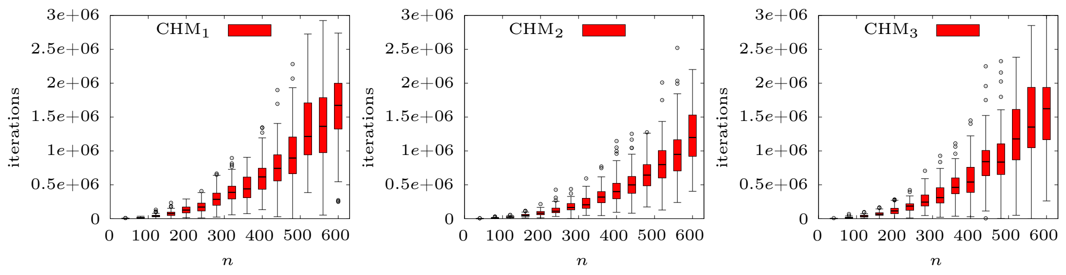


(f) medians, deterministic

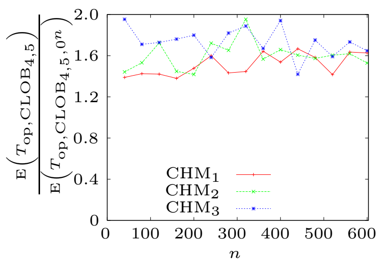
Figure 5.5.: Empirical data from 100 runs for TrailingOnes′.



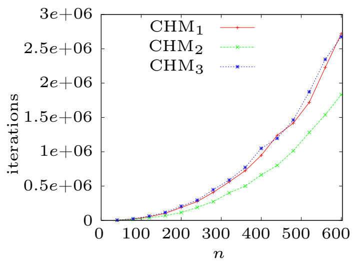
(a) random initialization



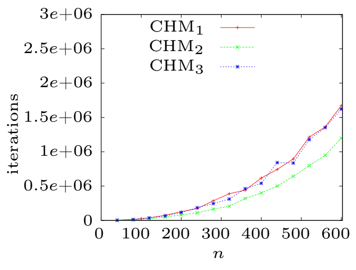
(b) deterministic initialization



(c) observed quotients

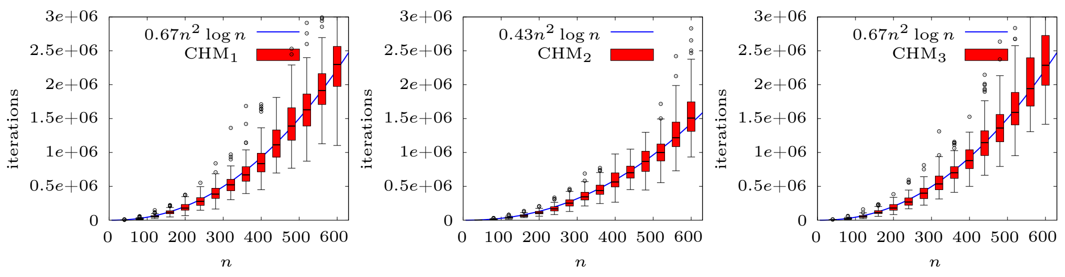


(d) medians, random

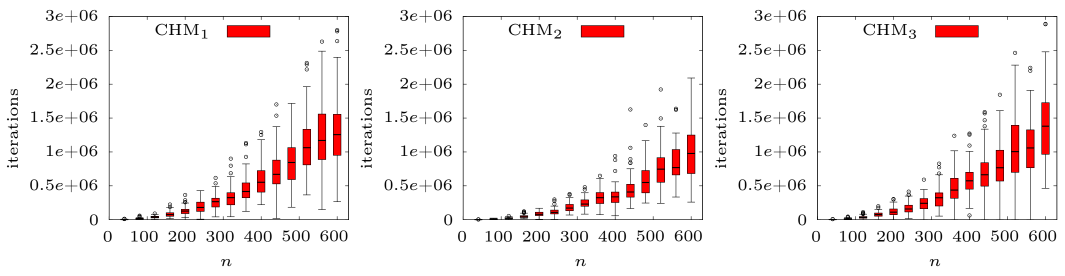


(e) medians, deterministic

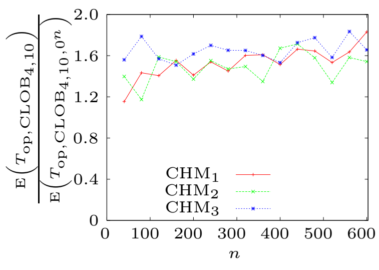
Figure 5.6.: Empirical data from 100 runs for $\mathrm{CLOB}_{4,5}$.



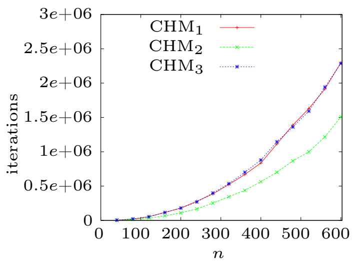
(a) random initialization



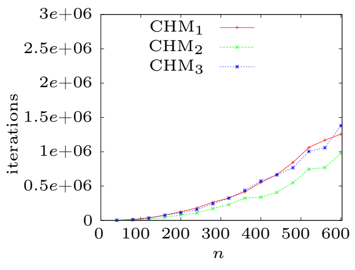
(b) deterministic initialization



(c) observed quotients

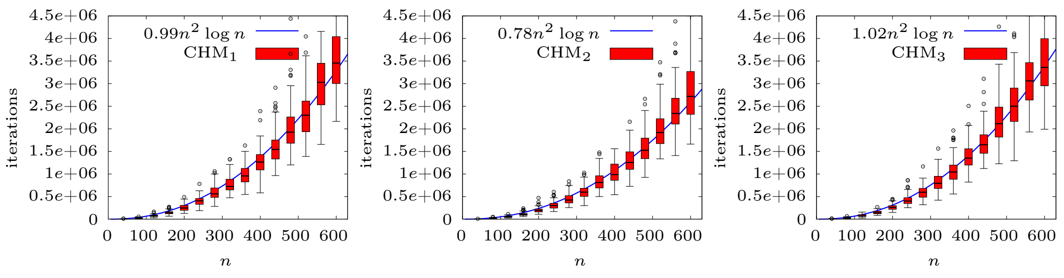


(d) medians, random

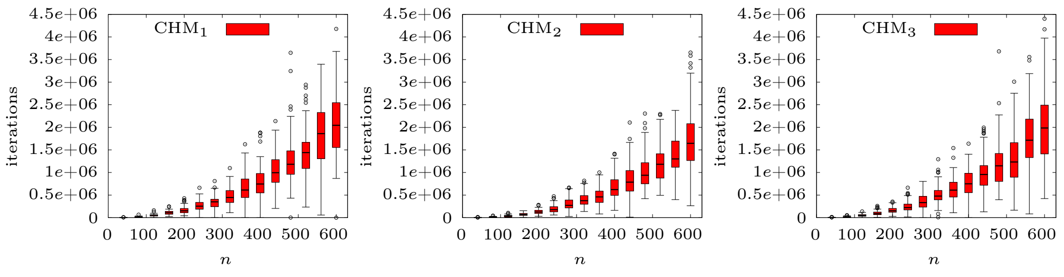


(e) medians, deterministic

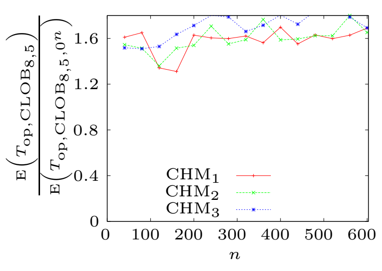
Figure 5.7.: Empirical data from 100 runs for CLOB$_{4,10}$.

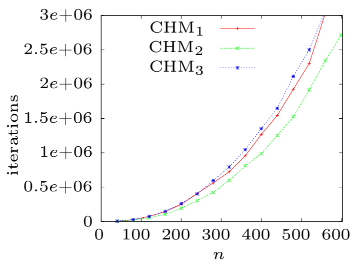


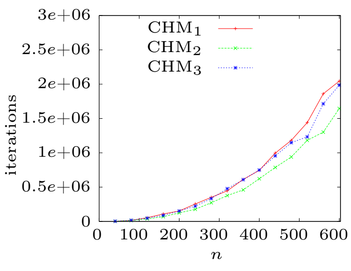
(a) random initialization



(b) deterministic initialization

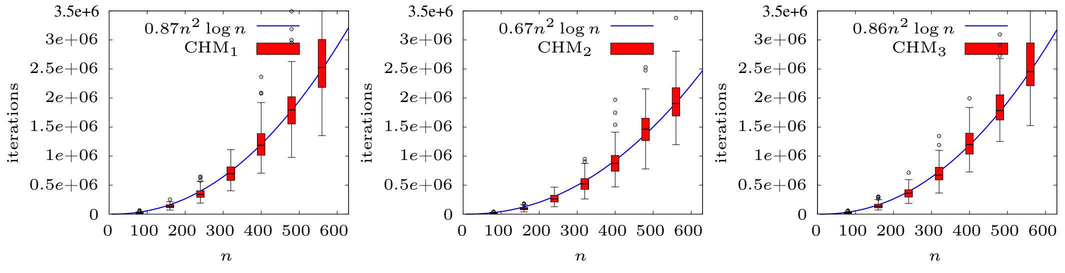




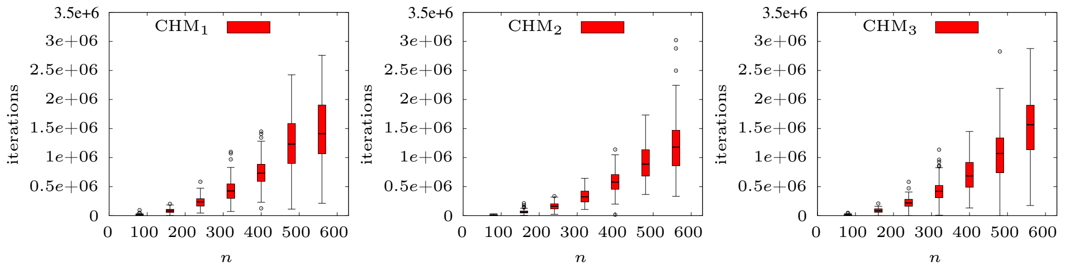


(c) observed quotients (d) medians, random (e) medians, deterministic

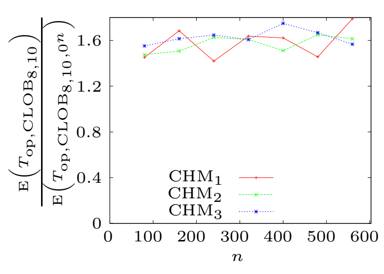
Figure 5.8.: Empirical data from 100 runs for $\mathrm{CLOB}_{8,5}$.



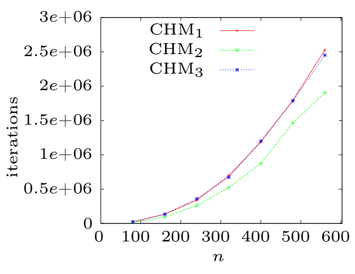
(a) random initialization



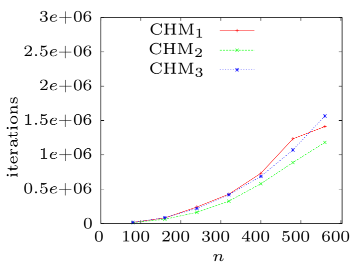
(b) deterministic initialization



(c) observed quotients



(d) medians, random



(e) medians, deterministic

Figure 5.9.: Empirical data from 100 runs for $\mathrm{CLOB}_{8,10}$.

6. On the Effect of Large Mutation Probabilities When Optimizing Monotone Functions

In the previous two chapters, we have investigated several aspects of common immune-inspired mutation operators, namely inversely fitness-proportional and somatic contiguous hypermutations. While we showed on which kind of problems contiguous hypermutations excel over standard bit mutations used in evolutionary algorithms, we have seen that parametrization is crucial for inversely fitness-proportional mutation probabilities and that larger collections of search points can help when using certain concrete instantiations of this kind of operator. However, when looking closer at the results for inversely fitness-proportional mutations one recognizes that these kind of mutations yield polynomial optimization times on OneMax only when the used parametrization results in some mutation probability $\Theta(1/n)$, at least in relevant parts of the search space.

In Section 4.3.2 of this thesis, we have discussed an algorithm, where each search point in the collection of search points is equipped with an individual mutation probability. We have shown that, under certain circumstances, these mutation probabilities are all $\Theta(1/n)$ but with noticeable differences between them. Note, that the mutation probability is not fixed during a run of the algorithm. To be more precise, the constant factor hidden in $\Theta(1/n)$ lies in an interval $[c_1, c_2]$ for constants $c_2 > c_1 > 0$. Moreover, in large parts of the search space, we have $c_1 > 1$. It is an interesting question what effects such a range of mutation probabilities can have. Moreover, it was an open problem if there exists a function where increasing the mutation probability from $1/n$ to c/n for a constant $c > 1$ increases the optimization time by more than a constant factor. In this chapter, we address both questions by considering the (1+1) EA with mutation probability $p(n) = c/n$ for some fixed constant c.

While for OneMax it was known before that a mutation probability $p(n) = c/n$ for all fixed constants c yields polynomial optimization time (Droste et al. 2002), this observation raises (amongst others) the question what effect larger mutation probability can have in general, in particular when only increasing the mutation probability by a constant factor. In order to address this question we consider Algorithm 3.1 with $\mu = 1$ using standard bit mutations from Algorithm 3.5 with some fixed mutation probability $p(n) = c/n$ ($c > 0$ constant). Note, that this is a general version of the (1+1) EA (see Definition 3.1). To simplify notation, we use this term in the following.

We analyze the effect when using values much larger than 1 for the constant c in the mutation probability on the class of strictly monotone pseudo-Boolean functions defined formally later. These functions have the property that whenever only 0-bits are changed to 1, the function value strictly increases. This implies that all these functions are easy to optimize for randomized local search, only flipping single bits. One may expect that they are also easy to optimize for the (1+1) EA with mutation probability $\Theta(1/n)$. However, the choice of the constant c in the mutation probability can make a decisive difference. This is the first time that we observe that a constant factor change of the mutation probability changes the optimization time by more than constant factors. Since in artificial immune systems typically even larger mutation probabilities are used, this result is relevant for hypermutations and shows serious drawbacks of this kind of mutation operator.

We start our considerations by presenting known results for different classes of pseudo-Boolean functions. Afterwards, we first investigate the optimization time for small mutation probabilities. Finally, the main part of this chapter deals with the result on larger mutation probabilities. The result of this chapter is based on the work done in Doerr et al. (2010a, 2011).

6.1. Known Results for Different Classes of Functions

Before we come to our main result, we present previous related work with respect to classes of pseudo-Boolean functions. We consider three different, important and well-known classes: linear functions, monotone functions, and unimodal functions. We discuss relations between these classes and summarize known results on the optimization time of the (1+1) EA on these classes.

A pseudo-Boolean function f is called *linear* if it can be written as a weighted sum of the bits in a search point. We define such functions formally in Definition 6.1.

Definition 6.1. *The class of* linear *pseudo-Boolean functions $f\colon \{0,1\}^n \to \mathbb{R}$ is defined by*

$$\textsc{Lin} := \left\{ f\colon \{0,1\}^n \to \mathbb{R} \mid f(x) = w_n + \sum_{i=0}^{n-1} w_i x[i], w_0, \ldots, w_{n-1}, w_n \in \mathbb{R} \right\}.$$

Note, that when considering the (1+1) EA on linear functions, we can w. l. o. g. assume, that all weights are non-negative, since the (1+1) EA is symmetric with respect to 0-bits and 1-bits and thus, we can easily replace $x[i]$ by $1 - x[i]$ (Droste et al. 2002). We consider a subset of $\textsc{Lin}$, where all weights except w_n are strictly positive. Note, that having weights of 0 can only simplify optimization and thus, does not influence upper bounds. This yields the function class $\textsc{Lin}'$ defined in the following.

Definition 6.2. *The restricted class of* linear *pseudo-Boolean functions $f\colon \{0,1\}^n \to \mathbb{R}$ is defined by*

$$\textsc{Lin}' := \left\{ f\colon \{0,1\}^n \to \mathbb{R} \mid f(x) = w_n + \sum_{i=0}^{n-1} w_i x[i], w_0, \ldots, w_{n-1} \in \mathbb{R}^+, w_n \in \mathbb{R} \right\}.$$

The most simple linear function, where all weights are set to 1, is OneMax (Definition 4.1), which we considered in the previous chapters. Another intensively studied linear function is BinVal, denoting the decimal value that is represented by the respective bit string. Since we need BinVal later in our definition in Section 6.3, we give a formal definition in the following.

Definition 6.3. *For $n \in \mathbb{N}$ and $x \in \{0,1\}^n$, the function* $\textsc{BinVal}\colon \{0,1\}^n \to \mathbb{R}$ *is defined by*

$$\textsc{BinVal}(x) = \sum_{i=0}^{n-1} 2^{n-i-1} x[i].$$

As $2^{n-i-1} > \sum_{j=i+2}^{n} 2^{n-j}$, the bit value of some bit i dominates the effect of all bits $i+1,\ldots,n-1$ on the function value. It is known that the optimization time of the (1+1) EA with $p(n) = c/n$ on restricted linear functions (Lin′) is $\Theta(n \log n)$ for all constants $c > 0$ (Doerr and Goldberg 2010).

For *strictly monotone* pseudo-Boolean functions (usually called simply *monotone* in the following) it holds that flipping only 0-bits to 1-bits strictly increases the function value of the search point. We define this function class in Definition 6.4

Definition 6.4. *The class of* (strictly) monotone *pseudo-Boolean functions $f\colon \{0,1\}^n \to \mathbb{R}$ is defined by*

$$\textsc{Mon} := \{f\colon \{0,1\}^n \to \mathbb{R} \mid \forall x, y \in \{0,1\}^n\colon (x > y) \Rightarrow (f(x) > f(y))\},$$

where $x > y \Leftrightarrow \forall i \in \{0,\ldots,n-1\}\colon x[i] \geq y[i]$, $\exists j \in \{0,\ldots,n-1\}\colon x[j] > y[j]$.

Observe that the condition in Definition 6.4 is equivalent to $f(x) < f(y)$ for all x and y such that x and y only differ in exactly one bit and this bit has value 1 in y. Clearly, the all-ones bit string 1^n is the unique global optimum for a monotone function. It is easy to see that monotone functions are just the ones where a simple coupon collector argument (Lemma B.17) shows that random local search, i. e., the mutation operator flipping exactly one bit, finds the optimum in time $\Theta(n \log n)$. This is due the fact that flipping a single bit from 0 to 1 implies that the fitness strictly increases and thus this 1-bit will never be lost again. However, for the (1+1) EA things are much more involved. We consider the optimization time of the (1+1) EA with mutation probability $p(n) = c/n$ for different constant values of $c > 0$ within this chapter.

If for two arbitrary search points x, y neither $x < y$ nor $y < x$ holds, we say that x and y are *incomparable*. This happens if and only if there are two different bit positions i and j such that $x[i] = 0$ but $y[i] = 1$ and $x[j] = 1$ but $y[j] = 0$. Note that monotonicity does not impose any restrictions on the fitness values of x and y. In other words, if f is monotone then any of the following cases can occur: $f(x) > f(y)$, $f(x) = f(y)$, or $f(x) < f(y)$. When defining a monotone function, we can choose any of the above cases for f, as long as no monotonicity constraint involving other search points is violated. This in particular indicates that the class of monotone functions contains much more complex functions than linear functions.

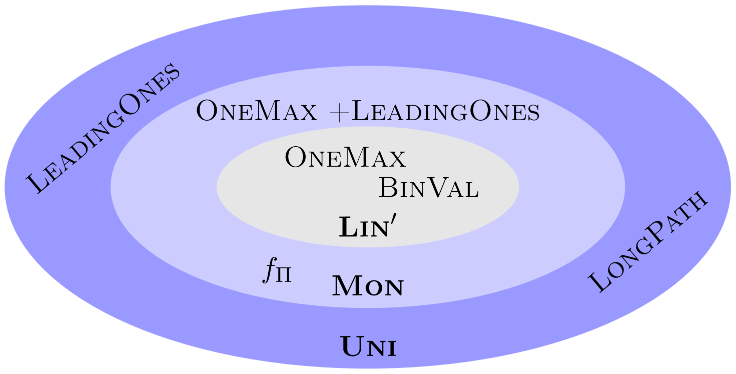


Figure 6.1.: Classes of pseudo-Boolean functions.

Finally, *unimodal* functions comprise all pseudo-Boolean functions where each non-optimal point has at least one better Hamming neighbor (see Definition 6.5). Droste et al. (2006) showed that the black-box complexity of unimodal functions with $b(n)+n$ different function values is $\Omega\big(b(n)/\log^2(b(n))\big)$ in the case $b(n)=2^{o(n)}$. Earlier, for the (1+1) EA a weaker lower bound was established using a concrete objective function, namely long k-paths due to Horn et al. (1994) and Rudolph (1996). It was shown that the (1+1) EA does with high probability not find the global optimum within $2^{\sqrt{n}}$ iterations (Droste et al. 1998). Our definition in the next section follows a similar idea. Moreover, we make use of a long k-path function when considering aging in Chapter 8.

Definition 6.5. *The class of* unimodal *pseudo-Boolean functions* $f\colon\{0,1\}^n\to\mathbb{R}$ *is defined by*

$$\begin{aligned}\textsc{Uni} := \Big\{ f\colon \{0,1\}^n \to \mathbb{R} \mid \forall x \in \{0,1\}^n\colon\ &(\forall y \in \{0,1\}^n\colon H(x,y)=1 \Rightarrow f(x) \ge f(y)) \\ &\Rightarrow (f(x) = \max\{f(z) \mid z \in \{0,1\}^n\}) \Big\}.\end{aligned}$$

Note that every linear function with strictly positive weights is a strictly monotone function as flipping only 0-bits to 1 strictly increases the fitness. Also recall that every monotone function is unimodal since for each non-optimal search point, i. e., for each $x \neq 1^n$ we can flip exactly one 0-bit and get a Hamming neighbor y with $f(y) > f(x)$. However, much stronger than for unimodal functions, we not only require that each non-optimal x has a Hamming neighbor with better f-value, but we even ask that this holds for all Hamming neighbors that have an additional 1-bit. Contrary to the long k-path function there is always a short path of at most n search points with increasing f-value connecting a search point with the optimum. We visualize the relation of the considered function classes and well-known representatives in Figure 6.1. Note, that f_Π is the monotone function, we are defining later in Section 6.3.

Summarizing what we know so far, we see that for linear functions the (1+1) EA with mutation probability $p(n) = c/n$ achieves a polynomial optimization time for all constant values $c > 0$, while on unimodal functions, we have exponential optimization time with high probability for all c. For monotone functions, i. e., the class 'inbetween' linear and unimodal functions, the dependency on c was not known. In the remainder of this chapter, we will see that in this case the choice of c is crucial for the efficiency of the considered algorithm and that the (1+1) EA is not able to optimize monotone functions in polynomial optimization time in general if c is too large.

6.2. Optimizing Monotone Functions Using Small Mutation Rates

For the (1+1) EA, the difficulty of monotone functions strongly depends on the mutation probability $p(n)$. We are interested in mutation probabilities $p(n) = c/n$ for some constant $c > 0$. Recall, that we denote by $\mathrm{mut}(x)$ the bit string that results from a mutation of x and by x^+ the search point that results from a mutation of x and a subsequent selection. If c is a constant with $c < 1$, on average, less than one bit flips in a single mutation. If this is a 1-bit we have $f(x) > f(\mathrm{mut}(x))$ and $x = x^+$ holds. Otherwise, $f(x^+) > f(x)$ holds and we accept this move. This way the number of 0-bits is quickly reduced to 0 and the unique global optimum is found. Using drift analysis this reasoning can easily be made precise.

Theorem 6.6 (Doerr et al. (2011)). *Let $c \in]0,1[$ be a constant. For every monotone function the expected optimization time of the* (1+1) EA *with mutation probability $p(n) = c/n$ is*

$$E\left(T_{(1+1)\ \mathrm{EA},\textsc{Mon}}\right) = \Theta(n \log n).$$

Proof. For the lower bound, we use exactly the same arguments as in Theorem 4.2 (Droste et al. 2002). For the upper bound we employ multiplicative drift analysis (Theorem B.23).

We consider the distance measure $d\colon \{0,1\}^n \to \{0,1,\dots,n\}$ with $d(x) := |x|_0$. Let x denote the current bit string of the (1+1) EA. In order to derive an upper bound on $\mathrm{E}\left(d(x^+) \mid x\right)$ we distinguish the two cases $d(x) = d(x^+)$ and $d(x) \neq d(x^+)$. By the law of total probability (Lemma B.11)

$$\begin{aligned}\mathrm{E}\left(d(x^+) \mid x\right) = \mathrm{Prob}\left(d(x) = d(x^+)\right) \cdot d(x)&\\ + \mathrm{Prob}\left(d(x) \neq d(x^+)\right) \cdot \mathrm{E}\left(d(x^+) \mid x, d(x) \neq d(x^+)\right)&\end{aligned}$$

holds. In the case $d(x) \neq d(x^+)$ the bit string x^+ replaces x. This can only be the case if at least one bit flipped from 0 to 1. Each of the remaining $n-1$ bits flips with probability c/n and may increase the distance by 1. This yields

$$\mathrm{E}\left(d(x^+) \mid x, d(x) \neq d(x^+)\right) \leq d(x) - 1 + (n-1) \cdot \frac{c}{n} \leq d(x) - (1-c)$$

as upper bound and we obtain

$$\begin{aligned}\mathrm{E}\left(d(x^+) \mid x\right) &\leq \mathrm{Prob}\left(d(x) = d(x^+)\right) \cdot d(x) + \mathrm{Prob}\left(d(x) \neq d(x^+)\right) \cdot (d(x) - (1-c)) \\ &= d(x) - \mathrm{Prob}\left(d(x) \neq d(x^+)\right) \cdot (1-c).\end{aligned}$$

A sufficient condition for $d(x) \neq d(x^+)$ is that exactly one of the $d(x)$ bits with value 0 in x flips and all other bits remain unchanged. This event has probability

$$\binom{d(x)}{1} \cdot \frac{c}{n} \cdot \left(1 - \frac{c}{n}\right)^{n-1} \leq d(x) \cdot \frac{ce^{-c}}{n}$$

and leads to

$$\mathrm{E}\left(d(x^+) \mid x\right) \leq d(x) - d(x) \cdot \frac{ce^{-c}}{n} \cdot (1-c) = d(x) \cdot \left(1 - \frac{ce^{-c}(1-c)}{n}\right)$$

as upper bound. Applying the drift theorem (Theorem B.23) with $\delta = (ce^{-c}(1-c))/n$, $c_{\min} = 1$, and $c_{\max} = n$, we obtain

$$\frac{n}{ce^{-c}(1-c)} \cdot \ln(1+n)) = O(n \log n)$$

as upper bound on the expected optimization time. □

The proof of the lower bound is not restricted to $c \in]0,1[$. For any constant $c > 0$ the number of steps considered, i. e., $c^{-1}(n-c)\ln n$, is $\Omega(n \log n)$. This implies the following corollary.

Corollary 6.7 (Doerr et al. (2011)). *Let $c > 0$ be a constant. For every monotone function the expected optimization time of the* (1+1) EA *with mutation probability $p(n) = c/n$ is*

$$E\left(T_{(1+1)\text{ EA},\textsc{Mon}}\right) = \Omega(n \log n).$$

The proof of the upper bound in Theorem 6.6 breaks down for $c = 1$. In this case the drift in the number of 1-bits can be bounded pessimistically by a model due to Jansen (2007) where we consider a random process that mutates x to y with mutation probability $p(n) = 1/n$ and replaces x by y if either $x \leq y$ holds or we have neither $x \leq y$ nor $y \leq x$ but $|y|_1 < |x|_1$ holds.

The model is pessimistic in the following sense. Every mutation that flips only 0-bits to 1-bits is guaranteed to lead to an improvement in the function value for every monotone function and is accepted in this model, too. For the analysis of the model as well as the (1+1) EA on a monotone function a drift analysis could be employed using the number of 0-bits as drift function. With respect to this drift function the model is more pessimistic than any monotone function since each mutation that potentially decreases the number of 1-bits in a monotone function is accepted. This cannot happen for a monotone function. To see this consider for example $n = 4$ and the following sequence of bit strings: $s_0 = 0111$, $s_1 = 1100$, $s_2 = 0001$, $s_3 = 0011$. In the pessimistic model we could have s_0, s_1, s_2, s_3, s_0 as sequence of current bit strings. This cannot be the case for the (1+1) EA with any monotone function since $f(s_2) < f(s_3) < f(s_0)$ holds by definition of monotonicity. Having s_0, s_1 as sequence of current bit strings implies $f(s_1) \geq f(s_0)$ and since $f(s_0) > f(s_2)$ we cannot have s_2 as next current bit string. Thus, the pessimistic model allows for cycles that are not possible for the (1+1) EA with any monotone function.

Using the number of 0-bits as drift function, the worst case model can yield an upper bound for the expected optimization time of the (1+1) EA with mutation probability $p(n) = 1/n$ on monotone functions. This way, we obtain the upper bound of $O(n^{3/2})$ for $p(n) = 1/n$.

Theorem 6.8 (Jansen (2007))**.** *For every monotone function the expected optimization time of the* (1+1) EA *with mutation probability* $p(n) = 1/n$ *is*

$$E\left(T_{(1+1)\text{ EA},\textsc{Mon}}\right) = O\Big(n^{3/2}\Big).$$

6.3. On the Effects of Large Mutation Rates

The main result of this chapter is that using mutation probability $p(n) = c/n$, where c is a sufficiently large constant, optimization of monotone functions can become very difficult for the (1+1) EA. This is the first result where increasing the mutation probability by a constant factor increases the optimization time from polynomial to exponential with overwhelming probability. We start with a formal statement of this result.

Theorem 6.9 (Doerr et al. (2011))**.** *For every constant* $c \geq 16$ *the following holds. For all* $n \in \mathbb{N}$*, there exist a monotone function* $f\colon \{0,1\}^n \to \mathbb{N}$ *and a constant* $\kappa > 0$ *such that, with probability* $1 - 2^{-\Omega(n)}$*, the* (1+1) EA *with mutation probability* $p(n) = c/n$ *does not optimize* f *within* $2^{\kappa n}$ *iterations.*

The remainder of this chapter is devoted to the formal proof of Theorem 6.9. In Section 6.3.1 we first present a family of fitness functions of which many have the desired property. In Section 6.3.2, we describe why these functions are difficult to optimize.

We make use of the following notation. For $x = (x[0], \dots, x[n-1])$ let $Z(x)$ describe the positions of all 0-bits in x, i. e., $Z(x) := \{0 \leq i \leq n-1 \mid x[i] = 0\}$ and $|Z(x)| = |x|_0$. For $k \in \mathbb{N}$ let $[k] := \{1, 2, \dots, k\}$ and $[k]_0 = \{0\} \cup [k]$. For a set $I = \{i_1, i_2, \dots, i_\ell\} \subseteq [n-1]_0$ we write $x_{|I} := x_{i_1} x_{i_2} \cdots x_{i_\ell}$ for the sub-string of x with the bits selected by I. To simplify notation we assume that any time we consider some $r \in \mathbb{R}_0^+$ but in fact need some $r' \in \mathbb{N}_0$ we assume that r is silently replaced by $\lfloor r \rfloor$ or $\lceil r \rceil$ as appropriate.

6.3.1. Definition and Properties of a Difficult to Optimize Monotone Function

The main idea is the definition of a kind of long path function, similar to the work by Horn et al. (1994). This definition uses the probabilistic method (Alon and Spencer

2000). To the best of our knowledge, this is the first time that problem instances are defined this way in the theory of randomized search heuristics. In their work, Horn et al. (1994) defined a path of Hamming neighbors of exponential length. The probability of taking a shortcut by mutation, that is, jumping forward a long distance on the path, is very small as many bits have to flip simultaneously. All points that are not on the path have an unfavorable fitness, such that the considered algorithm is forced to follow the path to the end.

Here, we also have an exponentially long path such that shortcuts can only be taken if a large number of bits flip simultaneously, a very unlikely event. The definition is complicated by the fact that the function needs to be monotone. Hence we cannot forbid leaving the path by giving the boundary of the path an unfavorable fitness. We solve this problem, roughly speaking, by implementing the path on a level of bit strings having similar numbers of 1-bits. Monotonicity only forbids leaving the level to strings having fewer 1-bits without differences in the 0-bits. The path is 'broad' in that sense that the algorithm can gather some additional 1-bits without leaving the path. The crucial part of our definition is setting up the function in such a way that, in spite of monotonicity, not too many 1-bits are collected.

Our path will be located in a region where the number of 1-bits is already fairly large. If the mutation probability c/n is large, it is likely that more 1-bits are flipped to 0 than 0-bits are flipped to 1. So, when mutating a point on the path it is likely that we have a net loss in terms of the number of 1-bits. This effect becomes more pronounced the more 1-bits the mutated search point has. The behavior of the (1+1) EA of course depends on whether such a net loss will be accepted. Monotonicity requires that whenever only 1-bits are flipped to 0 then the fitness must decrease. If at least one 0-bit is flipped together with at least one 1-bit, the two search points are incomparable. Hence, even for a monotone fitness function such a transition might be accepted. Our long path function is defined in such a way that operations leading to a net loss of 1-bits when moving to an incomparable offspring are often accepted, while the current search point is on the path. This prevents the algorithm from gathering too many 1-bits and hence from leaving the path.

These considerations particularly apply to a subset of bits that we call a *window*. The precise subset determines the position on the long path; the set of bits in the window changes as the algorithm moves along on the path. More formally, for a subset of indices $B \subseteq [n-1]_0$ and for $x \in \{0,1\}^n$ the bits $x[i]$ with $i \in B$ are referred to as *window*. The bits $x[i]$ with $i \notin B$ are *outside the window*. Inside the window the function value is given by BinVal. The weights for BinVal are ordered differently for each window in order to avoid correlation between windows. The window is placed such that there is only a small number of 0-bits outside the window. Reducing the number of 0-bits outside causes the window to be moved. This is a likely event that happens frequently. However, we manage to define an exponentially long sequence of windows with the additional property that in order to come from one window to another window at large distance (in the sense of this sequence), a large number of bits needs to be flipped simultaneously. Since this is highly unlikely, it is very likely that the sequence of windows is followed, i. e., we do not jump from one window to another one at large distance. Thus, following the path takes, with overwhelming probability, an exponential number of steps. Droste et al. (1998) embed the long path into a unimodal function in a way that the (1+1) EA reaches the beginning of the path with probability close to 1. We adopt this technique and extend it to our monotone function.

The following Lemma 6.10 defines the sequence of windows of our function by defining the index sets B_i. Concrete values for the upcoming constants β and γ will be given later on in Theorem 6.15. The property that windows with large distance have large Hamming distance is formally stated as $|i-j| \geq \ell \Rightarrow |B_i \cap B_j| \leq \gamma\ell$ for $\ell = \Theta(n)$ and some constant $\gamma > 0$.

Lemma 6.10 (Doerr et al. (2011))**.** *Let $\beta, \gamma \in \mathbb{R}$ be constants with $\beta > 0$, $\gamma < 1$ and $\rho := \beta/(1-\beta) < \gamma < 2\rho$. Let $n \in \mathbb{N}$, $\ell := \beta n$ and $L := \lfloor \exp\big((\gamma-\rho)^2(1-\beta)n/6\big) \rfloor$. Finally, let $L' := L - \ell + 1$. Then there exist $b_1, b_2, \ldots, b_L \in [n-1]_0$ such that the following holds. Let $B_i := \{b_i, b_{i+1}, \ldots, b_{i+\ell-1}\}$ for all $i \in [L']$. Then*

(i) $|B_i| = \ell$ for all $i \in [L']$,

(ii) $|B_i \cap B_j| \leq \gamma\ell$ for all $i, j \in [L']$ such that $|i-j| \geq \ell$.

Proof. The proof invokes the probabilistic method (Alon and Spencer 2000), that is, we describe a way to randomly choose the b_i that ensures that properties (i) and (ii) hold with positive probability. This necessarily implies the existence of such a sequence $b_1, \ldots, b_L$.

Let the $b_1, b_2, \ldots, b_L$ be chosen uniformly at random subject to condition (i). More precisely, let $b_1 \in [n-1]_0$ be chosen uniformly at random. If $b_1, \ldots, b_{i-1}$ are already chosen, then choose b_i from $[n-1]_0 \setminus \{b_{\max\{1,i-\ell\}}, \ldots, b_{i-1}\}$ uniformly at random.

Let $i, j \in [L']$ with $i < j$ and $|i-j| \geq \ell$. By definition, the sets B_i and B_j do not share an index in $[L]$. Fix any outcome of B_i. For all $k \in \{0, \ldots, \ell-1\}$ let X_k be the indicator random variable for the event $b_{j+k} \in B_i$. Then $|B_i \cap B_j| = \sum_{k=0}^{\ell-1} X_k$. We have that, conditional on any outcomes of all other b_{j+k-t}, $t \in [j+k-1]$, the probability $\text{Prob}\,(X_k = 1)$ that $b_{j+k} \in B_i$ is at most $|B_i|/(n-\ell) = \beta/(1-\beta) = \rho$.

From this we first conclude that $E(|B_i \cap B_j|) \leq \rho\ell$. In addition, we may apply the Chernoff bound for moderately independent random variables from Lemma B.22 with the X_i^* being independent indicator variables taking the value one with probability $\beta/(1-\beta)$, and conclude via a simple Chernoff bound (Lemma B.21) that

$$\text{Prob}\,(|B_i \cap B_j| > \gamma\ell) \leq \text{Prob}\left(\sum_{k=0}^{\ell-1} X_k^* > \left(1 + \frac{\gamma-\rho}{\rho}\right) \cdot \rho\ell\right) \leq \exp\left(-\frac{\left(\frac{\gamma-\rho}{\rho}\right)^2 \rho\ell}{3}\right).$$

Since there are less than $(L')^2$ choices of (i,j), a simple union bound (Lemma B.10) yields

$$\operatorname{Prob}\left(\exists i,j \in [L']\colon (|i-j| \geq \ell) \wedge (|B_i \cap B_j| > \gamma\ell)\right) < (L')^2 \cdot \exp\left(-\frac{\left(\frac{\gamma-\rho}{\rho}\right)^2 \rho\ell}{3}\right) < 1. \qquad \square$$

One technical tool in the definition of the set of difficult monotone functions are (random) permutations of vectors that allow us to effectively reduce dependencies between bits. We define the notation for this tool in the following definition.

Definition 6.11 (Doerr et al. (2011)). *Let β, γ, ℓ, L, L', the b_i and B_i be as in Lemma 6.10. Let $\alpha \in \mathbb{R}$ with $0 < \alpha < \beta$. For $x \in \{0,1\}^n$ let $\mathcal{B}_x := \{i \in [L'] \mid |Z(x) \setminus B_i| \leq \alpha n\}$. Let $i_x^* := \max \mathcal{B}_x$, if $\mathcal{B}_x$ is non-empty. For $i \in [L']$ let $\pi^{(i)}$ be a permutation of B_i. Denote by $\Pi = (\pi^{(1)}, \ldots, \pi^{(L')})$ the sequence of these permutations. We use the short-hand $\pi^{(i)}(x)$ to denote the vector obtained from permuting the components of $(x[b_i], \ldots, x[b_{i+\ell-1}])$ according to $\pi^{(i)}$. Consequently, $\pi^{(i)}(x) = \left(x[\pi^{(i)}(b_i)], \ldots, x[\pi^{(i)}(b_{i+\ell-1})]\right)$.*

The following definition introduces a set of monotone functions most of which will turn out to be difficult to optimize. The definition assumes the sequence of windows B_i to be given. For $x \in \{0,1\}^n$ we say that some $i \in [L']$ is a *potential position* in the sequence of windows if the number of 0-bits outside the window B_i is limited by αn, $\alpha > 0$ some constant. The set of all potential positions is $\mathcal{B}_x$. We select the largest potential position i as actual position (and call it i_x^*) and have the function value for x depend mostly on this position. If no potential position i exists, we have not yet found the path of windows and lead the (1+1) EA towards it. If $i = L'$, i. e., the end of the path is reached, the (1+1) EA is lead towards the unique global optimum via OneMax.

Definition 6.12 (Doerr et al. (2011)). *Let β, γ, ℓ, L, L', the b_i and B_i be as in Lemma 6.10. Let α, $\mathcal{B}_x$, i_x^* be defined as in Definition 6.11. Let $\pi^{(i)}$ be any permutation of B_i. We use the notation, in particular Π, as introduced in Definition 6.11. Moreover, we use the definition of* OneMax *(Definition 4.1) and* BinVal *(Definition 6.3).*

We define $f_\Pi : \{0,1\}^n \to \mathbb{N}_0$ via

$$f_\Pi(x) := \begin{cases} \left|x_{|[n-1]_0 \setminus B_1}\right|_1 \cdot 2^n + \textsc{BinVal}\left(\pi^{\left(\left|x_{|[n-1]_0 \setminus B_1}\right|_1\right)}(x)\right), & \text{if } \mathcal{B}_x = \emptyset, \\ i_x^* \cdot 2^{2n} + \textsc{BinVal}\left(\pi^{(n+i_x^*)}(x)\right), & \text{if } \mathcal{B}_x \neq \emptyset \text{ and } n + i_x^* < L', \\ L \cdot 2^{3n} + |x|_1, & \text{otherwise.} \end{cases}$$

We state one observation concerning the function f_Π that is important in the following. It states that as long as the end of the path of windows is not found, the number of 0-bits outside is not only bounded by αn but equals αn. This property will be used later on to show that the window is moved frequently.

Lemma 6.13 (Doerr et al. (2011))**.** *Let $f_\Pi\colon \{0,1\}^n \to \mathbb{N}_0$ be as in Definition 6.12. Let $x \in \{0,1\}^n$ with $\mathcal{B}_x \neq \emptyset$ and $i_x^* = \max \mathcal{B}_x$. If $n + i_x^* < L'$, then $\left|Z(x) \setminus B_{i_x^*}\right| = \alpha n$.*

Proof. By assumption we have $n + i_x^* < L'$. We consider $B_{n+i_x^*+1}$ and see that the set coincides with $B_{n+i_x^*}$ in all but two elements: we have $B_{n+i_x^*} \setminus B_{n+i_x^*+1} = \{b_{n+i_x^*}\}$ and $B_{n+i_x^*+1} \setminus B_{n+i_x^*} = \{b_{n+i_x^*+\ell}\}$. Consequently, $|Z(x) \setminus B_{n+i_x^*}|$ and $|Z(x) \setminus B_{n+i_x^*+1}|$ differ by at most one. Thus, $|Z(x) \setminus B_{n+i_x^*}| < \alpha n$ implies $|Z(x) \setminus B_{n+i_x^*+1}| \leq \alpha n$ and we can replace i_x^* by $i_x^* + 1$. This contradicts $i_x^* = \max \mathcal{B}_x$. We have $|Z(x) \setminus B_{n+i_x^*}| \leq \alpha n$ by definition and thus $|Z(x) \setminus B_{n+i_x^*}| = \alpha n$ follows. □

Our first main claim is that f_Π is in fact monotone. This is not difficult, but might, due to the complicated definition of f_Π, not be obvious.

Lemma 6.14 (Doerr et al. (2011))**.** *For all Π as above, f_Π is monotone.*

Proof. Let $f := f_\Pi$. Let $x \in \{0,1\}^n$ and $j \in [n-1]_0$ such that $x[j] = 0$. Let $y \in \{0,1\}^n$ be such that $y[k] = x[k]$ for all $k \in [n-1]_0 \setminus \{j\}$ and $y[j] = 1 - x[j]$. That is, y is obtained from x by flipping the j-th bit (which is 0 in x) to one. To prove the lemma, it suffices to show $f(x) < f(y)$.

Let first $\mathcal{B}_x = \emptyset$. If $\mathcal{B}_y \neq \emptyset$ we have $f(x) < n \cdot 2^n + 2^n$ and $f(y) \geq 2^{2n}$ so $f(x) < f(y)$ follows. If $\mathcal{B}_y = \emptyset$ we have either $\left|x_{|[n-1]_0 \setminus B_1}\right|_1 < \left|y_{|[n-1]_0 \setminus B_1}\right|_1$ (in case $j \notin B_1$) or $\textsc{BinVal}(\pi^{(i)}(x)) < \textsc{BinVal}(\pi^{(i)}(y))$ (in case $j \in B_1$). In both cases, $f(x) < f(y)$ holds.

Now assume $\mathcal{B}_x \neq \emptyset$ and $n + i_x^* < L'$. By definition $\mathcal{B}_x \subseteq \mathcal{B}_y$, hence $i_y^* \geq i_x^*$. If $i_y^* = i_x^*$, we conclude with Lemma 6.13 that $j \in B_{i_x^*}$, and $f(y) > f(x)$ follows from $\textsc{BinVal}(\pi^{(n+i_y^*)}(y)) = \textsc{BinVal}(\pi^{(n+i_x^*)}(y)) > \textsc{BinVal}(\pi^{(n+i_x^*)}(x))$. If $i_y^* > i_x^*$, then $f(y) > f(x)$. In all other cases, $f(x) = L2^{3n} + |x|_1$ and $f(y) = L2^{3n} + |y|_1$, hence $f(y) > f(x)$. □

6.3.2. A Lower Bound on the Optimization Time

By means of the function defined in Definition 6.12, we are now ready to prove a lower bound on the optimization time for the class of pseudo-Boolean monotone functions. We start with the concrete statement we want to prove. This result shows that if f is chosen randomly (according to the definition described above), then the (1+1) EA with overwhelming probability needs an exponential time to find the optimum. Clearly, this implies that there exists a particular function f, that is, a choice of Π, such that the EA faces these difficulties. This is Theorem 6.9. In fact, there is even an exponential number of functions for which this holds. The parameters α, β, and γ in Theorem 6.15 were chosen to obtain a small constant in the threshold $16/n$ for the mutation probability.

Theorem 6.15 (Doerr et al. (2011))**.** *Consider the* (1+1) EA *with mutation probability c/n for a constant $c \geq 16$ on the function $f := f_\Pi$ from Definition 6.12 where Π is chosen uniformly at random and the parameters are chosen according to $\beta := 1/5$, $\gamma := 30/113$, and $\alpha := 3/(400c)$. There is a constant $\kappa > 0$ such that with probability $1 - 2^{-\Omega(n)}$ the* (1+1) EA *needs at least $2^{\kappa n}$ iterations to optimize f.*

The proof of Theorem 6.15 is long and technical. We prove the claim in 3 steps. We first show that it is very unlikely to take large shortcuts once the path is reached, implying that the algorithm is forced to follow the path. Afterwards, we make use of drift arguments in order to show that there is always a linear fraction of 0-bits within the current window until the end of the path is reached or an exponential number of iterations have passed. The proof of this part is separated into two parts, one dealing with the case of a moving window and one, where the window stays put. Finally we show that we hit the beginning of the path starting from a random initialization with overwhelming probability. Putting these results together then proves Theorem 6.15.

We remark, that the proof of Theorem 6.15 and the statements below will all be carried out in a parameterized fashion as above. Thus, we actually prove that Theorem 6.15 holds whenever the following conditions are met.

$$\begin{aligned} 0 < \eta_{\text{lb}} &< \eta_{\text{ub}} < 1/2 \\ 0 < \alpha &< \beta < \gamma < 1 \\ \eta_{\text{lb}} - \frac{\alpha}{\beta} &> \gamma \\ \frac{\beta}{1-\beta} &< \gamma < \frac{2\beta}{1-\beta} \\ c\beta &> \frac{2-2\eta_{\text{ub}}}{1-2\eta_{\text{ub}}} \\ \alpha &< \frac{1-2\eta_{\text{ub}}}{3c} \end{aligned} \tag{6.1}$$

It is easy to check that all conditions are fulfilled by the settings from Theorem 6.15, i. e., whenever $c \geq 16$, $\beta := 1/5$, $\gamma := 30/113$, $\alpha := 3/(400c)$, $\eta_{\text{lb}} := 30/112$, and $\eta_{\text{ub}} := 30/111$.

Unlikeliness of Shortcuts

We consider the (1+1) EA with mutation probability c/n and say that the (1+1) EA is on *level* i_x^* if x is the current search point. We also speak of *phase* i_x^* as the random time until the (1+1) EA increases its current level. Note that many phases can be empty. $B_{i_x^*}$ is called the *current window* of bits in situations where we are looking at a trajectory of these sets and want to emphasize that the bits we are considering might change over time.

The main observation for our analysis is that the current window typically contains at least $\eta_{\text{lb}}\beta n$ 0-bits for some positive constant η_{lb}. This property is maintained even during an exponential number of iterations, with overwhelming probability. Under this condition, the probability of increasing the current level i_x^* by a large value is very small. Intuitively speaking, the reason for this is that the sets B_i only have a small intersection and many bits have to change in order to move from $B_{i_x^*}$ to some set B_j with $j \gg i_x^*$.

To be more precise, in general, every two sets B_i, B_j with $|i-j| \geq \ell$ only intersect in at most $\gamma\beta n$ bits. So $\eta_{\text{lb}}\beta n$ 0-bits in B_i imply at least $\eta_{\text{lb}}\beta n - \gamma\beta n$ 0-bits outside of B_j.

For j to become the new window, however, at most αn 0-bits outside of B_j are allowed. By choice of α, β, and γ, moving from B_i to B_j requires a linear number of 0-bits in B_i to flip to 1 if $j > \beta n$. The described mutation has probability $n^{-\Omega(n)}$. Hence, even when considering an exponential period of time, with overwhelming probability the (1+1) EA in each iteration only makes progress at most βn on the path. This is made precise in the following lemma.

Lemma 6.16 (Doerr et al. (2011)). *Let $0 < \alpha < \beta < \gamma$ and $0 < \eta_{\mathrm{lb}}$ be constants such that $\eta_{\mathrm{lb}} - \alpha/\beta > \gamma$ and $\beta/(1-\beta) < \gamma < 2\beta/(1-\beta)$. Let f_Π, with respect to α, β, and γ, be constructed as in Definition 6.12, for arbitrary Π. Let $c > 0$ be a constant and let x be the current search point of the* (1+1) EA *with mutation probability $p(n) = c/n$ optimizing f_Π. Assume that $\mathcal{B}_x \neq \emptyset$ and $B_{i_x^*}$ contains at least $\eta_{\mathrm{lb}}\beta n$ 0-bits. Then the probability that the* (1+1) EA *increases the level i_x^* by more than βn in one iteration is at most $n^{-\Omega(n)}$.*

Proof. Since $\eta_{\mathrm{lb}}\beta n > \alpha n + \gamma\beta n$ it holds that $B_{i_x^*}$ contains more than $\alpha n + \gamma\beta n$ 0-bits. Recall that $|B_{i_x^*} \cap B_j| \leq \gamma\beta n$ for all $j \geq i_x^* + \beta n$. Thus, there are more than αn 0-bits outside of B_j. This implies, by the definition of $\mathcal{B}_x$ that a necessary condition for increasing i_x^* to any value $j \geq i_x^* + \beta n$ is thus that one mutation decreases the number of 0-bits in $B_{i_x^*}$ to a value below or equal to $\alpha n + \gamma\beta n$. This is a decrease of at least $\eta_{\mathrm{lb}}\beta n - \alpha n - \gamma\beta n =: \kappa n$ bits for some constant $0 < \kappa < 1$. The probability of flipping at least κn bits simultaneously is at most $\binom{n}{\kappa n} \cdot (c/n)^{\kappa n} \leq c^{\kappa n} \cdot 1/(\kappa n)! = n^{-\Omega(n)}$. $\square$

One conclusion from this lemma is that, with overwhelming probability, the (1+1) EA follows the path given by the sets B_i without jumping from one window to another one at large distance. More precisely, each phase increases the current level by at most βn with overwhelming probability. This will establish the claimed bound on the optimization time.

Proving an Invariance Property on the Number of 0-bits in the Current Window

Both after a typical initialization, when $\mathcal{B}_x = \emptyset$, and afterwards, when $\mathcal{B}_x \neq \emptyset$ and $n + i_x^* < L'$, we have the following situation. There is a window of bits ($B_{i_x^*}$ if i_x^* is defined and B_1 otherwise) such that the fitness of the search points depends mainly on the BinVal function inside the window. Moreover, the fitness is always increased in case the mutation decreases the number of 0-bits outside the window. If $\mathcal{B}_x = \emptyset$ this is due to the term $\left|x_{|[n]\setminus B_1}\right|_1 \cdot 2^n$ in the fitness function and otherwise it is because the current i_x^*-value has increased. The gain in fitness is so large that it dominates any change of the bits inside the window.

We claim that with this construction it is very likely that the current window always contains at least $\eta_{\mathrm{lb}}\beta n$ 0-bits, where η_{lb} is some positive constant. This is proven by showing that in case the number of 0-bits in the window is in the interval $[\eta_{\mathrm{lb}}\beta n, \eta_{\mathrm{ub}}\beta n]$, $0 < \eta_{\mathrm{lb}} < \eta_{\mathrm{ub}} < 1/2$ constant, then there is a tendency ('drift') to increase the number of 0-bits again. Applying a drift theorem by Oliveto and Witt (2010) yields that even in an exponential number of iterations the probability that the number of 0-bits in the window decreases below $\eta_{\text{lb}}\beta n$ is exponentially small. We first elaborate on why the drift on the number of 0-bits holds. It will be bounded from below by positive constants in two cases: either the current level i^*_x remains fixed in one iteration or it is increased. We start with the latter case and give a lower bound for the number of 0-bits in the current window. At the end of this section, we apply the drift theorem by Oliveto and Witt (2010) to prove the claim.

Before formulating the main statements of this section, we need to introduce some notations. For any x let $x_B := x_{|B_{i^*_x}}$ denote the substring of x induced by $B_{i^*_x}$, i. e., the substring in the current window. Recall that $|x_B|_0$ denotes the number of 0-bits in the current window. That is, $|x_B|_0 = |\{j \in \{b_{i^*_x}, \dots, b_{i^*_x+\ell-1}\} \mid x[j] = 0\}|$. For the sake of readability we write $\left|x^+_B\right|_0$ instead of $|(x^+)_B|_0$ for the number of 0-bits of x^+ in its window.

A prerequisite for this theorem is that the number of 0-bits in the current window increases in expectation, when the number of 0-bits is in a certain interval. We choose the interval $[\eta_{\text{lb}}\beta n, (\eta_{\text{lb}} + \eta_{\text{ub}})/2 \cdot \beta n]$, where $0 < \eta_{\text{lb}} < \eta_{\text{ub}} < 1/2$, but establish lower bounds for the drift with respect to a larger interval $[\eta_{\text{lb}}\beta n, \eta_{\text{ub}}\beta n]$. The larger interval will be used later on when proving that the after initialization the (1+1) EA finds the start of the path (Lemma 6.21).

Invariance for Sliding Windows

We first consider the case where the current level is increased, i. e., a transition from i^*_x to $i^*_{x^+}$ with $i^*_x < i^*_{x^+} < L'$ happens. Note that here, we deal with the case $i^*_x \neq i^*_{x^+}$ and thus, $B_{i^*_x} \neq B_{i^*_{x^+}}$ and $x^+_B \neq x_B$ holds. We show that in this situation we have a drift in the number of 0-bits within the current window that is bounded below by a positive constant. Due to the transition it is not sufficient to only consider changes within the current window. Furthermore, transitions are often triggered by changes outside the current window. Thus, we assume a form of a 'global' view and take into account both the changes within the current window as well as changes outside the current window.

If a mutation decreases the number of 0-bits outside the window, the bits inside the window are subject to random, unbiased mutations. Hence, if the number of 0-bits is at most $\eta_{\text{ub}}\beta n$, the expected number of bits flipping from 1 to 0 is larger than the expected number of bits flipping from 0 to 1. Note that a mutation flipping 0-bits to 1 outside the window and flipping 1-bits to 0 inside the window creates an incomparable offspring. If the mutation probability is large enough, the net gain of 0-bits inside the window makes up for the 0-bits lost outside the window. So we have a net gain in 0-bits in expectation, with regard to the whole bit string. Note that the window is moved during such a mutation. As by Lemma 6.13 the number of 0-bits outside the window is fixed to αn, we have a net gain in 0-bits for the window, regardless of its new position. We formalize this within the next lemma.

Lemma 6.17 (Doerr et al. (2011)). *Let $0 < \alpha < \beta < 1$, $0 < \eta_{\text{ub}} < 1/2$, and c be constants such that $\alpha < (1-2\eta_{\text{ub}})/3c$ and $c\beta > (2-2\eta_{\text{ub}})/(1-2\eta_{\text{ub}})$. Let n be sufficiently large and let f, with respect to α and β, be constructed as in Theorem 6.15. Let x be the* *current search point of the* (1+1) EA *with mutation probability* $p(n) = c/n$ *maximizing* f. *We denote by* $\bar{A}$ *the event that a transition from level* i^*_x *to* $i^*_{x^+}$ *with* $i^*_x < i^*_{x^+} < L'$ *occurs in an iteration of the* (1+1) EA *maximizing* f. *Assume* $|x_B|_0 \leq \eta_{\mathrm{ub}}\beta n$.

Then, there is a constant $\delta > 0$ *such that the drift in the number of* 0*-bits is at least* δ, *i. e.,* $E\left(|x^+_B|_0 - |x_B|_0 \mid \bar{A}\right) \geq \delta$.

Proof. Let $\overline{B}_{i^*_x} = [n-1]_0 \backslash B_{i^*_x}$, the indices not contained in the current window, and $x_{\overline{B}} := x|_{\overline{B}_{i^*_x}}$ the corresponding induced substring of x. Analogously, we define $x^+_{\overline{B}} := x|_{\overline{B}_{i^*_{x^+}}}$. Due to Lemma 6.13 we have $\left|x_{\overline{B}}\right|_0 = \left|x^+_{\overline{B}}\right|_0 = \alpha n$. The main part of the proof is to derive a lower bound on $\mathrm{E}\left(|x^+_B|_0 \mid \bar{A}\right)$. Afterwards we show that this bound together with the given prerequisites on α, β, c and $|x_B|_0$ yields a positive drift in the number of 0-bits.

It is easy to see that, conditional on $\bar{A}$, the expected number of 0-bits in the new window $B_{i^*_{x^+}}$ after a transition from i^*_x to $i^*_{x^+}$ can be derived as the difference of the expected number of 0-bits in the current window $B_{i^*_x}$ after mutation and the expected amount of 0-bits lost outside the current window due to mutation:

$$\mathrm{E}\left(|x^+_B|_0 \mid \bar{A}\right) = \mathrm{E}\left(|\mathrm{mut}(x_B)|_0 \mid \bar{A}\right) - \mathrm{E}\left(\left|x_{\overline{B}}\right|_0 - \left|\mathrm{mut}(x_{\overline{B}})\right|_0 \mid \bar{A}\right) \tag{6.2}$$

We derive bounds for both parts of eq. (6.2) separately. We start with an upper bound on the expected number of 0-bits in the current window after mutation, i. e., $\mathrm{E}\left(|\mathrm{mut}(x_B)|_0 \mid \bar{A}\right)$ by the following case distinction.

In the first case, the transition happens independently of the change in the window. This case happens with probability $\Omega(1)$ as a 1-bit mutation of one of the αn 0-bits outside the current window suffices. In this situation, the expected number of 0-bits in the window is independent of $\bar{A}$ and thus, can be easily calculated as follows.

$$\begin{aligned}\mathrm{E}\left(|\mathrm{mut}(x_B)|_0\right) &= \left(1 - \frac{c}{n}\right)|x_B|_0 + \frac{c}{n}|x_B|_1 = |x_B|_0 - \frac{c}{n}|x_B|_0 + \frac{c}{n}|x_B|_1 \\ &= |x_B|_0 + \frac{c}{n}\left(\beta n - 2|x_B|_0\right) = |x_B|_0 + c\beta - \frac{2c|x_B|_0}{n}\end{aligned}$$

For the second case, i. e., if the mutation within the current window has influence on the transition performed, we have to be more careful as the expected number of 0-bits within the window is no longer independent of $\bar{A}$. However, before the mutation the leftmost bit in the current window is 0. Otherwise, the next window position would also be a potential, higher window position. This contradicts the definition of i^*_x. If this leftmost 0-bit is flipped a transition is performed. Furthermore, it is necessary to flip this leftmost 0-bit if the mutation within the window is supposed to have influence on the transition performed. The probability of flipping this bit is c/n and thus the probability for this case is at most c/n.

We bound the contribution of this case in a pessimistic way. Similar to Lemma 6.16 we see that the number of bits flipping in one single iteration is at most $O(\log n)$ with probability $1 - n^{-\omega(1)}$. Furthermore, the contribution is at most βn otherwise. Altogether, this yields a contribution to the expected value of at most

$$\frac{c}{n} \cdot \left(\left(1 - n^{-\omega(1)}\right) \cdot \log n + n^{-\omega(1)} \cdot \beta n\right) = O\left(\frac{\log n}{n}\right)$$

and we get the following lower bound on the expected number of 0-bits within the current window after mutation.

$$\mathrm{E}\left(\left|\mathrm{mut}(x_B)\right|_0 \mid \bar{A}\right) \geq \left|x_B\right|_0 + c\beta - \frac{2c\left|x_B\right|_0}{n} - O\left(\frac{\log n}{n}\right) \tag{6.3}$$

The second part of eq. (6.2), i. e., the expected loss of 0-bits outside the current window due to mutation, is more difficult. For the sake of readability, let $k := \left|x_{\overline{B}}\right|_0 - \left|\mathrm{mut}(x_{\overline{B}})\right|_0$ denote the loss of 0-bits outside the current window due to mutation. Then, we are searching for $\mathrm{E}\left(k \mid \bar{A}\right)$. We distinguish two cases. If $k > 0$ we definitely observe a transition and accept the new search point. If $k \leq 0$ a transition does not necessarily occur. For $k < 0$ the new search point is only accepted if this is the case. For $k = 0$ the search might also be accepted depending on the changes within the current window. We see that $\mathrm{E}(k) \leq \mathrm{E}\left(k \mid \bar{A}\right) \leq \mathrm{E}(k \mid k > 0)$ holds.

Let Z_0 be the number of 0-bits flipping to one and Z_1 the number of 1-bits flipping to zero. This yields $\mathrm{E}(k \mid k > 0) = \mathrm{E}(Z_0 - Z_1 \mid Z_0 > Z_1)$. Moreover, we observe that

$$\begin{aligned}
&\sum_{j=i+1}^{\alpha n} j\mathrm{Prob}(Z_0 = j) \\
=\ & (i+1)\mathrm{Prob}(Z_0 = i+1) + (i+1)\mathrm{Prob}(Z_0 = i+2) + \cdots + (i+1)\mathrm{Prob}(Z_0 = \alpha n) \\
&\qquad + \mathrm{Prob}(Z_0 = i+2) \qquad + \cdots + \mathrm{Prob}(Z_0 = \alpha n) \\
&\qquad\qquad + \mathrm{Prob}(Z_0 = i+3) + \cdots + \mathrm{Prob}(Z_0 = \alpha n) \\
&\qquad\qquad\qquad \ddots \\
&\qquad\qquad\qquad\qquad + \mathrm{Prob}(Z_0 = \alpha n) \\
=\ & (i+1)\mathrm{Prob}(Z_0 > i) + \mathrm{Prob}(Z_0 > i+1) + \mathrm{Prob}(Z_0 > i+2) + \cdots \\
&\quad + \mathrm{Prob}(Z_0 > \alpha n - 1) \\
=\ & (i+1)\mathrm{Prob}(Z_0 > i) + \sum_{j=i+1}^{\alpha n} \mathrm{Prob}(Z_0 > j)
\end{aligned} \tag{6.4}$$

holds. Then, with $\left|x_{\overline{B}}\right|_0 = \alpha n$ and $\left|x_{\overline{B}}\right|_1 = (1 - \alpha - \beta)n$ we can rewrite the expected value sought as follows.

$$\begin{aligned}
\mathrm{E}(Z_0 - Z_1 \mid Z_0 > Z_1) &= \sum_{i=0}^{(1-\alpha-\beta)n} \mathrm{Prob}(Z_1 = i) \cdot \mathrm{E}(Z_0 - Z_1 \mid Z_0 > Z_1 \text{ and } Z_1 = i) \\
= \sum_{i=0}^{(1-\alpha-\beta)n} &\mathrm{Prob}(Z_1 = i) \cdot (\mathrm{E}(Z_0 \mid Z_0 > i) - \mathrm{E}(Z_1 \mid Z_0 > Z_1 \text{ and } Z_1 = i))
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{i=0}^{(1-\alpha-\beta)n} \operatorname{Prob}(Z_1 = i) \cdot \left(\operatorname{E}(Z_0 \mid Z_0 > i) - i\right) \\
&= \sum_{i=0}^{(1-\alpha-\beta)n} \operatorname{Prob}(Z_1 = i) \cdot \left(\sum_{j=0}^{\alpha n} j \cdot \operatorname{Prob}(Z_0 = j \mid Z_0 > i) - i \right) \\
&= \sum_{i=0}^{(1-\alpha-\beta)n} \operatorname{Prob}(Z_1 = i) \cdot \left(\sum_{j=0}^{\alpha n} j \cdot \frac{\operatorname{Prob}(Z_0 = j \text{ and } Z_0 > i)}{\operatorname{Prob}(Z_0 > i)} - i \right) \\
&= \sum_{i=0}^{(1-\alpha-\beta)n} \operatorname{Prob}(Z_1 = i) \cdot \left(\frac{\sum_{j=i+1}^{\alpha n} j \cdot \operatorname{Prob}(Z_0 = j)}{\operatorname{Prob}(Z_0 > i)} - i \right) \\
&\overset{(6.4)}{=} \sum_{i=0}^{(1-\alpha-\beta)n} \operatorname{Prob}(Z_1 = i) \cdot \left(\frac{(i+1) \cdot \operatorname{Prob}(Z_0 > i) + \sum_{j=i+1}^{\alpha n} \operatorname{Prob}(Z_0 > j)}{\operatorname{Prob}(Z_0 > i)} - i \right) \\
&= \sum_{i=0}^{(1-\alpha-\beta)n} \operatorname{Prob}(Z_1 = i) \cdot \left(1 + \frac{\sum_{j=i+1}^{\alpha n} \operatorname{Prob}(Z_0 > j)}{\operatorname{Prob}(Z_0 > i)} \right). \qquad (6.5)
\end{aligned}
$$

It is easy to see the following estimations for the probabilities used above.

$$
\begin{aligned}
\operatorname{Prob}(Z_0 > j) &\le \binom{\alpha n}{j+1} \cdot \left(\frac{c}{n}\right)^{j+1} \\
\operatorname{Prob}(Z_0 > i) &\ge \binom{\alpha n}{i+1} \cdot \left(\frac{c}{n}\right)^{i+1} \cdot \left(1 - \frac{c}{n}\right)^{\alpha n - i - 1} \\
\operatorname{Prob}(Z_1 = i) &= \binom{(1-\alpha-\beta)n}{i} \cdot \left(\frac{c}{n}\right)^{i} \cdot \left(1 - \frac{c}{n}\right)^{(1-\alpha-\beta)n - i}
\end{aligned}
$$

Plugging these inequalities into eq. (6.5) yields the following expression for the expected loss of 0-bits outside the current window due to mutation.

$$
\begin{aligned}
\operatorname{E}\left(k \mid \bar{A}\right) \le \sum_{i=0}^{(1-\alpha-\beta)n} &\binom{(1-\alpha-\beta)n}{i} \cdot \left(\frac{c}{n}\right)^{i} \cdot \left(1 - \frac{c}{n}\right)^{(1-\alpha-\beta)n - i} \\
&\cdot \left(1 + \frac{\sum_{j=i+1}^{\alpha n} \binom{\alpha n}{j+1} \left(\frac{c}{n}\right)^{j+1}}{\binom{\alpha n}{i+1} \left(\frac{c}{n}\right)^{i+1} \left(1 - \frac{c}{n}\right)^{\alpha n - i - 1}} \right)
\end{aligned}
$$

$$\overset{(B.13)}{=} 1 + \sum_{i=0}^{(1-\alpha-\beta)n} \binom{(1-\alpha-\beta)n}{i} \cdot \left(\frac{c}{n}\right)^i \cdot \left(1-\frac{c}{n}\right)^{(1-\alpha-\beta)n-i} \cdot \frac{\sum_{j=i+1}^{\alpha n} \binom{\alpha n}{j+1} \left(\frac{c}{n}\right)^{j+1}}{\binom{\alpha n}{i+1} \left(\frac{c}{n}\right)^{i+1} \left(1-\frac{c}{n}\right)^{\alpha n-i-1}} \tag{6.6}$$

We start with the second part of this term and derive the following lower bound.

$$\begin{aligned}
&\frac{\sum_{j=i+1}^{\alpha n} \binom{\alpha n}{j+1} \left(\frac{c}{n}\right)^{j+1}}{\binom{\alpha n}{i+1} \left(\frac{c}{n}\right)^{i+1} \left(1-\frac{c}{n}\right)^{\alpha n-i-1}} = \frac{\sum_{j=i+1}^{\alpha n} \frac{(\alpha n)!}{(j+1)!(\alpha n-j-1)!} \left(\frac{c}{n}\right)^{j+1} \frac{(i+1)!(\alpha n-i-1)!}{(\alpha n)!} \left(\frac{n}{c}\right)^{i+1}}{\left(1-\frac{c}{n}\right)^{\alpha n-i-1}} \\
&= \frac{1}{\left(1-\frac{c}{n}\right)^{\alpha n-i-1}} \sum_{j=i+1}^{\alpha n} \left(\frac{c}{n}\right)^{j-i} \frac{(i+1)!}{(j+1)!} \frac{(\alpha n-i-1)!}{(\alpha n-j-1)!} \\
&= \frac{1}{\left(1-\frac{c}{n}\right)^{\alpha n-i-1}} \sum_{j=i+1}^{\alpha n} \left(\frac{c}{n}\right)^{j-i} \cdot \frac{\alpha n-i-1}{i+2} \cdot \frac{\alpha n-i-2}{i+3} \cdot \dots \cdot \frac{\alpha n-j}{j+1} \\
&\leq \frac{1}{\left(1-\frac{c}{n}\right)^{\alpha n-i-1}} \sum_{j=i+1}^{\alpha n} \left(\frac{c}{n}\right)^{j-i} \left(\frac{\alpha n}{i+1}\right)^{j-i} = \frac{1}{\left(1-\frac{c}{n}\right)^{\alpha n-i-1}} \sum_{j=i+1}^{\alpha n} \left(\frac{c\alpha}{i+1}\right)^{j-i} \\
&= \frac{1}{\left(1-\frac{c}{n}\right)^{\alpha n-i-1}} \sum_{j=1}^{\alpha n-i} \left(\frac{c\alpha}{i+1}\right)^{j} \leq \frac{1}{\left(1-\frac{c}{n}\right)^{\alpha n-i-1}} \cdot \frac{\frac{c\alpha}{i+1}}{1-\frac{c\alpha}{i+1}} \leq \frac{1}{\left(1-\frac{c}{n}\right)^{\alpha n-i-1}} \cdot \frac{c\alpha}{1-c\alpha}
\end{aligned}$$

Remember that we assume $\alpha < (1-2\eta_{\text{ub}})/(3c)$ and thus $c\alpha < 1/3$ holds. Therefore, we can further simplify the above inequality by using the following simple estimation.

$$\begin{aligned}
\left(1-\frac{c}{n}\right)^{\alpha n-i-1} &\geq \left(1-\frac{c}{n}\right)^{\alpha n} = \left(1-\frac{c}{n}\right)^{\left(\frac{n}{c}-1\right)c\alpha} \left(1-\frac{c}{n}\right)^{c\alpha} \\
&\geq e^{-c\alpha} \left(1-\frac{c}{n}\right)^{c\alpha} = \left(\frac{1-\frac{c}{n}}{e}\right)^{c\alpha}
\end{aligned}$$

Plugging all this into eq. (6.6) yields a lower bound on the expected value sought.

$$\begin{aligned}
&\mathrm{E}\left(k \mid \bar{A}\right) \\
&\leq 1 + \sum_{i=0}^{(1-\alpha-\beta)n} \binom{(1-\alpha-\beta)n}{i} \cdot \left(\frac{c}{n}\right)^i \cdot \left(1-\frac{c}{n}\right)^{(1-\alpha-\beta)n-i} \cdot \left(\frac{e}{1-\frac{c}{n}}\right)^{c\alpha} \cdot \frac{c\alpha}{1-c\alpha} \\
&= 1 + \left(\frac{e}{1-\frac{c}{n}}\right)^{c\alpha} \cdot \frac{c\alpha}{1-c\alpha}
\end{aligned} \tag{6.7}$$

We are now able to put the results from eq. (6.3) and eq. (6.7) together to get a lower bound on eq. (6.2).

$$\begin{aligned}\mathrm{E}\left(\left|x_B^+\right|_0 \mid \bar{A}\right) =\ & \mathrm{E}\left(\left|\mathrm{mut}(x_B)\right|_0 \mid \bar{A}\right) - \mathrm{E}\left(\left|x_{\overline{B}}\right|_0 - \left|\mathrm{mut}(x_{\overline{B}})\right|_0 \mid \bar{A}\right)\\ \geq\ & |x_B|_0 + c\beta - \frac{2c\,|x_B|_0}{n} - O\left(\frac{\log n}{n}\right) - \left(1 + \left(\frac{e}{1-\frac{c}{n}}\right)^{c\alpha}\frac{c\alpha}{1-c\alpha}\right)\end{aligned}$$

With $|x_B|_0 \leq \eta_{\mathrm{ub}}\beta n$, this yields the following lower bound on the drift in the number of 0-bits.

$$\begin{aligned}\mathrm{E}\left(\left|x_B^+\right|_0 - |x_B|_0 \mid \bar{A}\right) \geq\ & c\beta - \frac{2c\,|x_B|_0}{n} - O\left(\frac{\log n}{n}\right) - \left(1 + \left(\frac{e}{1-\frac{c}{n}}\right)^{c\alpha}\frac{c\alpha}{1-c\alpha}\right)\\ \geq\ & c\beta(1-2\eta_{\mathrm{ub}}) - O\left(\frac{\log n}{n}\right) - 1 - \left(1-\frac{c}{n}\right)^{-c\alpha}\frac{e^{c\alpha}c\alpha}{1-c\alpha}\ .\end{aligned}$$

Now,

$$\left(1-\frac{c}{n}\right)^{-c\alpha} = \left(1+\frac{c}{n-c}\right)^{c\alpha} = 1 + O(1/n).$$

As the factor $(e^{c\alpha}c\alpha)/(1-c\alpha)$ is constant, we have $O(1/n)\cdot(c\alpha)/(1-c\alpha) = O(1/n)$ and thus, this term can be absorbed by the $O((\log n)/n)$-term. This results in the lower bound

$$\begin{aligned}& c\beta(1-2\eta_{\mathrm{ub}}) - O\left(\frac{\log n}{n}\right) - 1 - \frac{e^{c\alpha}c\alpha}{1-c\alpha}\\ \geq\ & c\beta(1-2\eta_{\mathrm{ub}}) - O\left(\frac{\log n}{n}\right) - 1 - \frac{3e^{1/3}}{2}\cdot c\alpha\ ,\end{aligned}\tag{6.8}$$

where we have again used $c\alpha < 1/3$. Combining the preconditions $c\beta > (2-2\eta_{\mathrm{ub}})/(1-2\eta_{\mathrm{ub}})$ and $\alpha < (1-2\eta_{\mathrm{ub}})/(3c)$ implies that

$$c\beta > \frac{1+(1-2\eta_{\mathrm{ub}})}{1-2\eta_{\mathrm{ub}}} > \frac{1+3c\alpha}{1-2\eta_{\mathrm{ub}}}\ .$$

Plugging this into eq. (6.8) yields

$$1 + 3c\alpha - O\left(\frac{\log n}{n}\right) - 1 - \frac{3e^{1/3}}{2}\cdot c\alpha = c\alpha\left(3 - \frac{3e^{1/3}}{2}\right) - O\left(\frac{\log n}{n}\right)$$

and this is bounded from below by a positive constant δ for sufficiently large values of n, which concludes the proof. □

Invariance for Non-Sliding Windows

In case the number of 0-bits outside the window remains put, acceptance depends on a BinVal instance on the bits inside the window. For BinVal accepting the result of a mutation is completely determined by the flipping bit with the largest weight. In an accepted step, this bit must have flipped from 0 to 1. All bits with smaller weights have no impact on acceptance and therefore are subject to random, unbiased mutations. If, among all bits with smaller weights, there is a sufficiently small rate of 0-bits, more bits will flip from 1 to 0 than from 0 to 1. In this case, we again obtain a net increase in the number of 0-bits in the window, in expectation. Here we again require a large mutation probability since every increase of BinVal implies that one 0-bit has been lost and a surplus of flipping 1-bits has to make up for this loss. This surplus must be generated by flipping 1-bits inside the window that have a small weight. Recall that the window only represents a β-fraction of all bits in the bit string. So, the mutation probability has to be large enough such that the expected number of flipping bits among the mentioned bits is still large enough to make up for the lost bit.

For a fixed BinVal instance the bits tend to develop correlations between bit values and weights over time; bits with large weights are more likely to become 1 than bits with small weights. This development is disadvantageous since the above argument relies on many 1-bits with small weights. In order to break up these correlations the definition of the considered function class is based on a sequence of random permutations. Whenever a window is moved a new permutation from the sequences is used. This way we consider BinVal on random permutations of bit strings and eliminate the correlation between the position in the bit string and the value of the bit

New random instances are applied quickly. If $\mathcal{B}_x = \emptyset$ and, by Lemma 6.13, also if $n+i_x^* < L'$ we have exactly αn 0-bits outside the current window and every mutation that flips exactly one of these bits leads to a new BinVal instance. Since this happens with probability $\Omega(1)$, this frequently breaks up correlations and prevents the algorithm from gathering 1-bits at bits inside the window with large BinVal-weights. Pessimistically dealing with bits that have been touched by mutation while optimizing the same BinVal instance, a positive expected increase in the number of 0-bits can be shown.

In the following we show that, whenever the number of 0-bits in the current window is in the interval $[\eta_{\text{lb}}\beta n, \eta_{\text{ub}}\beta n]$, we observe a drift towards more 0-bits. This is formalized in the following lemma. Note that, here, we always deal with the case that $i_x^* = i_{x^+}^*$.

Lemma 6.18 (Doerr et al. (2011)). *Let $0 < \alpha < \beta < 1$, $0 < \eta_{\text{lb}} < \eta_{\text{ub}} < 1/2$, and c be constants such that $c\beta > (2 - 2\eta_{\text{ub}})/(1 - 2\eta_{\text{ub}})$. Let n be sufficiently large and let f, with respect to α and β, be constructed as in Definition 6.12. Let x be the current search point of the* (1+1) EA *with mutation probability $p(n) = c/n$ maximizing f. Assume $|x_B|_0 \in [\eta_{\text{lb}}\beta n, \eta_{\text{ub}}\beta n]$. We denote by A the event that the* (1+1) EA *maximizing f and starting in x does not leave the current level, i. e., $i_x^* = i_{x^+}^*$. Then, the following two statements hold.*

1. *For every constant $\varepsilon > 0$ the number of different bits that are flipped during phase i_x^* is at most $2\varepsilon\beta cn$, with probability $1 - e^{-\Omega(n)}$.*

2. *For small enough $\varepsilon > 0$, assuming that the event from* 1) *holds, there exists a constant $\delta > 0$ such that the drift in the number of* 0*-bits $E\left(|x_B^+|_0 - |x_B|_0 \mid A\right)$ is at least δ.*

The proof of this lemma will heavily depend on the drift in the number of 0-bits induced by the random BinVal within the current window. In the proof of Lemma 6.18, we will have to deal with variable lengths of the considered bit string. Therefore, the following auxiliary lemma is formulated for a bandwidth of possible bit string lengths. One precondition is that bit weights of BinVal are assigned uniformly at random. This is the case right after a new BinVal instance has been set.

Lemma 6.19 (Doerr et al. (2011)). *Let $0 \leq \varepsilon < \beta < 1$, $0 < \eta_{\text{lb}} < \eta_{\text{ub}} < 1/2$, and c be constants such that*

$$c(\beta - \varepsilon) - 2\eta_{\text{ub}} c\beta > 2 - 2\eta_{\text{ub}} \cdot \frac{\beta}{\beta - \varepsilon} \tag{6.9}$$

and

$$c(\beta - \varepsilon) \geq 1 \ . \tag{6.10}$$

Consider the (1+1) EA *with mutation probability $p(n) = c/n$ maximizing a* BinVal *function on $u \geq \beta n - \varepsilon n$ bits where the weight of the bits are chosen uniformly at random, without replacement, from $\{2^0, 2^1, \ldots, 2^{\beta n - 1}\}$. Let $\tilde{x}$ denote the current search point. If $|\tilde{x}|_0 \in [\eta_{\text{lb}} u, \eta_{\text{ub}} \beta n]$ then there exists a constant $\tilde{\delta} > 0$ such that the drift in the number of* 0*-bits is at least $\tilde{\delta}$, i. e., $E(|\tilde{x}^+|_0 - |\tilde{x}|_0) \geq \tilde{\delta}$.*

We remark that the expectation is drawn both with respect to the random assignment of the function weights as well as with respect to the position of the 0-bits of $\tilde{x}$.

Proof of Lemma 6.19. Recall that whenever $\tilde{x}^+ = \tilde{x}$ holds, we have $|\tilde{x}^+|_0 - |\tilde{x}|_0 = 0$. Thus, we are only interested in the case $\tilde{x}^+ \neq \tilde{x}$. Note that in this case, the definition of BinVal implies that the bit with the largest weight is one that flips from 0 to 1 as otherwise the (1+1) EA would not accept $\text{mut}(\tilde{x})$ as a new search point. For all other bits that are being flipped in this iteration, the direction of the flipping bit (i. e., whether the bit itself is a 0-bit flipping to 1 or a 1-bit flipping to 0) is random and does only depend on the shares of 0- and 1-bits. We formalize this in the following.

For the sake of readability, we make use of the following notations. For every $k \in \{0, \ldots, u\}$ we denote by p_k the probability that the (1+1) EA flips exactly k bits in the mutation step. Clearly,

$$p_k = \binom{u}{k} \cdot \left(\frac{c}{n}\right)^k \cdot \left(1 - \frac{c}{n}\right)^{u-k}$$

for $k \geq 1$ and $p_0 = (1 - c/n)^u$.

Let us, for the moment, assume that exactly $k \geq 1$ bits are flipped and let us consider the substring of the flipping bits only. If we remove from the substring the bit with the largest weight (which flips from 0 to 1), we get that the expected number of 0-bits in this reduced substring equals $(k-1) \cdot (|\tilde{x}|_0 - 1)/(u-1)$. Analogously, the expected number of 1-bits in the bit string equals $(k-1) - (k-1) \cdot (|\tilde{x}|_0 - 1)/(u-1)$. Recall that we have chosen the bits to exactly those which are flipped in the mutation step. We thus obtain for this specific setting that the expected difference of $|\tilde{x}^+|_0 - |\tilde{x}|_0$ equals

$$\begin{aligned}
\mathrm{E}\left(\left|\tilde{x}^+\right|_0 - |\tilde{x}|_0\right) &= \left((k-1) - (k-1)\cdot\frac{|\tilde{x}|_0 - 1}{u-1}\right) - (k-1)\cdot\frac{|\tilde{x}|_0 - 1}{u-1} - 1\\
&= k\left(1 - 2\cdot\frac{|\tilde{x}|_0 - 1}{u-1}\right) - \left(2 - 2\cdot\frac{|\tilde{x}|_0 - 1}{u-1}\right).
\end{aligned}$$

Now, for any such k, $\mathrm{E}\left(|\tilde{x}^+|_0 - |\tilde{x}|_0 \mid k \text{ bits flip}\right)$ equals the probability that the flipping bit with the largest weight flips from 0 to 1 (which occurs with probability $|\tilde{x}|_0/u$) times the drift, conditional on k bit flips. The latter equals

$$k\cdot\left(1 - 2\cdot\frac{|\tilde{x}|_0 - 1}{u-1}\right) - \left(2 - 2\cdot\frac{|\tilde{x}|_0 - 1}{u-1}\right)$$

as outlined above. Combining these observations, we gain

$$\mathrm{E}\left(\left|\tilde{x}^+\right|_0 - |\tilde{x}|_0\right) = \sum_{k=1}^{u} p_k\cdot\frac{|\tilde{x}|_0}{u}\cdot\left(k\left(1 - 2\cdot\frac{|\tilde{x}|_0 - 1}{u-1}\right) - \left(2 - 2\cdot\frac{|\tilde{x}|_0 - 1}{u-1}\right)\right).$$

Clearly, $\sum_{k=0}^{u} p_k = 1$ as we are dealing with a probability distribution. Thus, $\sum_{k=1}^{u} p_k = 1-(1-c/n)^u$. On the other hand, $\sum_{k=1}^{u} p_k k = (c/n)\cdot u$ as this sum equals the expected number of bit flips. By the choice of the parameters we have $\sum_{k=1}^{u} p_k k \geq c(\beta-\varepsilon)$. This yields

$$\begin{aligned}
&\mathrm{E}\left(\left|\tilde{x}^+\right|_0 - |\tilde{x}|_0\right)\\
&\geq \frac{|\tilde{x}|_0}{u}\cdot\left(\left(1 - 2\cdot\frac{|\tilde{x}|_0 - 1}{u-1}\right)\cdot c\cdot(\beta-\varepsilon) - \left(2 - 2\cdot\frac{|\tilde{x}|_0 - 1}{u-1}\right)\cdot\left(1-\left(1-\frac{c}{n}\right)^u\right)\right)\\
&\geq \eta_{\mathrm{lb}}\cdot\left(\left(1 - 2\cdot\frac{|\tilde{x}|_0 - 1}{u-1}\right)\cdot c\cdot(\beta-\varepsilon) - \left(2 - 2\cdot\frac{|\tilde{x}|_0 - 1}{u-1}\right)\right).
\end{aligned}$$

Since due to eq. (6.10) we have $c(\beta-\varepsilon)\geq 1$ we can use the estimation

$$\frac{|\tilde{x}|_0 - 1}{u-1} \leq \frac{|\tilde{x}|_0}{u} \leq \eta_{\mathrm{ub}}\cdot\frac{\beta n}{u} \leq \eta_{\mathrm{ub}}\cdot\frac{\beta}{\beta-\varepsilon},$$

such that

$$\begin{aligned}
&\left(1 - 2\cdot\frac{|\tilde{x}|_0 - 1}{u-1}\right)\cdot c\cdot(\beta-\varepsilon) - \left(2 - 2\cdot\frac{|\tilde{x}|_0 - 1}{u-1}\right)\\
&\geq \left(1 - 2\cdot\eta_{\mathrm{ub}}\cdot\frac{\beta}{\beta-\varepsilon}\right)\cdot c\cdot(\beta-\varepsilon) - \left(2 - 2\cdot\eta_{\mathrm{ub}}\cdot\frac{\beta}{\beta-\varepsilon}\right)\\
&= c\,(\beta-\varepsilon) - 2\eta_{\mathrm{ub}}c\beta - 2 + 2\eta_{\mathrm{ub}}\cdot\frac{\beta}{\beta-\varepsilon}\,. \qquad (6.11)
\end{aligned}$$

Since we are only dealing with constants, the precondition eq. (6.9) implies eq. (6.11) and hence $\mathrm{E}\left(|\tilde{x}^+|_0 - |\tilde{x}|_0\right)$ are both bounded from below by some positive constant $\tilde{\delta}$. ☐

We can now easily deduct Lemma 6.18.

Proof of Lemma 6.18. Let us assume that event A (as defined in the statement) holds. Thus, the acceptance of the mutated bit string $\mathrm{mut}(x)$ is fully determined by the random BinVal within the current window and we can restrict our attention to the current window.

We start with proving the first claim. Let $\varepsilon > 0$ be some constant. We first prove an auxiliary claim stating that with probability $e^{-\Omega(n)}$ the time $T_{i_x^*}$ until the (1+1) EA exits level i_x^* is at most εn, i. e., we can assume that phase i_{x^*} does not take longer than εn steps. We afterwards show how to derive the original claim.

By definition, the (1+1) EA exits level i_x^* if exactly one of the αn 0-bits outside the current window is flipped. Thus, the probability $\mathrm{Prob}\left(i_x^* \neq i_{x^+}^*\right)$ of exiting the current level in one step is at least

$$\alpha n \cdot \frac{c}{n} \cdot \left(1 - \frac{c}{n}\right)^{n-1} \geq c\alpha e^{-c} \left(1 - \frac{c}{n}\right)^{c-1} .$$

It follows that the probability of not exiting level i_x^* in εn steps is at most

$$\left(1 - c\alpha e^{-c}\left(1 - \frac{c}{n}\right)^{c-1}\right)^{\varepsilon n} \leq e^{-c\alpha e^{-c}\left(1-\frac{c}{n}\right)^{c-1}\varepsilon n} = e^{-\Omega(n)}.$$

Now, the expected number of bits that have been flipped in εn steps is at most $\varepsilon n \cdot \beta n \cdot (c/n) = \varepsilon\beta c n$. We apply a standard Chernoff bound (Lemma B.21), and obtain that the probability that more than $2\varepsilon\beta c n$ bits are flipped in εn steps is at most $e^{-(1/3)\cdot\varepsilon\beta c n}$ $= e^{-\Omega(n)}$.

We continue with the second claim. Therefore, assume that no more than $2\varepsilon\beta c n$ bits are being flipped during phase i_x^*. We note already here that we can conclude the following. The probability of flipping a bit in the current iteration that has already been flipped in a former iteration of phase i_x^* is at most $2\varepsilon\beta c n \cdot (c/n) = 2\epsilon\beta c^2$. That is, $\mathrm{Prob}\,(G \mid A) \leq 2\varepsilon\beta c^2$, where we denote by G the event that in the current iteration, the (1+1) EA flips a bit that has already been flipped in a former iteration of phase i_x^*. Clearly, by the law of total probability (Lemma B.11)

$$\begin{aligned}\mathrm{E}\left(|x_B^+|_0 - |x_B|_0 \mid A\right) = \mathrm{Prob}\,(G \mid A) \cdot \mathrm{E}\left(|x_B^+|_0 - |x_B|_0 \mid A \wedge G\right)& \\ + (1 - \mathrm{Prob}\,(G \mid A)) \cdot \mathrm{E}\left(|x_B^+|_0 - |x_B|_0 \mid A \wedge \bar{G}\right)& ,\end{aligned}$$

where $\bar{G}$ denotes the complementary event of G. Now, whenever G occurs, we adopt a worst case view by assuming that all bits flip in the wrong direction, i. e., from 0 to 1. For this purpose let us, for the moment, assume that G holds. In this case, at least one bit flips and we assume pessimistically that each of the flipping bits reduces the number of 0-bits by 1. Note that, given that one bit flips, the expected number of total bit flips in the current window equals $1 + (c/n) \cdot (\beta n - 1) < 1 + c\beta$. Thus, we can bound $\mathrm{E}\left(|x_B^+|_0 - |x_B|_0 \mid A \wedge G\right)$ from below by $-1 - c\beta$. Then, under our assumption, we get

$$\mathrm{Prob}\,(G \mid A)\, \mathrm{E}\left(|x_B^+|_0 - |x_B|_0 \mid A \wedge G\right) \geq 2\varepsilon\beta c^2(-1 - c\beta) .$$

We now need to derive bounds for the second summand. For this purpose, we apply Lemma 6.19. As we are conditioning on $\bar{G}$, we apply the auxiliary lemma with u denoting the number of bits that have not been flipped in any former iteration of phase i_x^*. Furthermore, we are only interested in the substring $\tilde{x}$ of $x_{|B}$ consisting of these u yet unflipped bits. As we have seen in the first part of this proof, with probability $1 - e^{-\Omega(n)}$, $u \geq \beta n - 2\varepsilon\beta cn$ holds. Also recall that $c\beta > (2 - 2\eta_{\text{ub}})/(1 - 2\eta_{\text{ub}})$ or, equivalently, $c\beta - 2\eta_{\text{ub}} c\beta > 2 - 2\eta_{\text{ub}}$. As β, c, and η_{ub} are constants and the inequality is strict, we can find some small enough $\varepsilon > 0$ such that the stronger statement

$$c(\beta - \varepsilon) - 2\eta_{\text{ub}} c\beta > 2 - 2\eta_{\text{ub}} \cdot \frac{\beta}{\beta - \varepsilon}$$

holds, fulfilling the precondition eq. (6.9) in Lemma 6.19. In addition, $c\beta > (2 - 2\eta_{\text{ub}}) \cdot (1 - 2\eta_{\text{ub}}) \geq 2$, hence $c(\beta - \varepsilon) \geq 1$ for small enough ε, fulfilling precondition eq. (6.10) in Lemma 6.19. Invoking Lemma 6.19 yields $\mathrm{E}\left(|x_B^+|_0 - |x_B|_0 \mid A \wedge \bar{G}\right) \geq \tilde{\delta}$ for some positive constant $\tilde{\delta}$.

Altogether we obtain

$$\mathrm{E}\left(|x_B^+|_0 - |x_B|_0 \mid A\right) \geq 2\varepsilon\beta c^2(-1 - c\beta) + (1 - 2\varepsilon\beta c^2)\tilde{\delta}\,.$$

Finally, we observe that we can choose ε small enough such that this term can be bounded from below by some positive constant δ, as claimed. □

Applying the Drift Theorem

Using the results of the previous sections, we are able to prove the claimed invariance property by means of the drift theorem due to Oliveto and Witt (2010) (see Theorem B.24).

Lemma 6.20 (Doerr et al. (2011)). *Let $0 < \alpha < \beta < 1$, $0 < \eta_{\text{lb}} < \eta_{\text{ub}} < 1/2$, γ, and c be constants such that $\beta/(1 - \beta) < \gamma < 2\beta/(1 - \beta)$, $c\beta > (2 - 2\eta_{\text{ub}})/(1 - 2\eta_{\text{ub}})$, and $\alpha < (1 - 2\eta_{\text{ub}})/(3c)$. Let f_Π, with respect to α and β, be constructed as in Definition 6.12, for Π chosen uniformly at random.*

Assume that for the current search point x of the (1+1) EA *with mutation probability $p(n) = c/n$ it holds $\mathcal{B}_x \neq \emptyset$ and the current window contains at least $(\eta_{\text{lb}} + \eta_{\text{ub}})/2 \cdot \beta n$ 0-bits. There is a constant $\kappa > 0$ such that with probability $1 - 2^{-\Omega(n)}$ in the following $2^{\kappa n}$ iterations the* (1+1) EA *always has at least $\eta_{\text{lb}}\beta n$ 0-bits in the current window or the end of the path is reached.*

Proof. First observe that the event described in the first statement of Lemma 6.18 occurs with probability $1 - 2^{-\Omega(n)}$. By the union bound, the probability that the event occurs within $2^{\kappa n}$ phases is still $1 - 2^{-\Omega(n)}$ if $\kappa > 0$ is a sufficiently small constant.

We apply the drift theorem (Theorem B.24) to a potential that reflects the number of 0-bits in the current window. Consider the interval $[\eta_{\text{lb}}\beta n, (\eta_{\text{lb}} + \eta_{\text{ub}})/2 \cdot \beta n]$ and observe that by assumption the algorithm starts with a potential of at least $(\eta_{\text{lb}} + \eta_{\text{ub}})/2 \cdot \beta n$. Using Lemma 6.18 with the condition from the first paragraph and Lemma 6.17, if the current potential is within the interval and the end of the path is not reached then the expected increase in the potential is bounded from below by a positive constant.

For $j \in \mathbb{N}_0$ the probability that the potential decreases by j is bounded from above by the probability that the (1+1) EA flips at least j bits. This probability is at most $\binom{n}{j}(c/n)^j \le c^j/(j!) \le (ec/j)^j \le 2^{-j} \cdot 2^{2ec}$, where the last estimation is trivial for $j \le 2ec$ and obvious otherwise. Applying Theorem B.24 with $\delta := 1$ and $r := 2^{2ec}$ yields that with overwhelming probability in $2^{\kappa n}$ iterations, if again κ is sufficiently small, the potential does not decrease below $\eta_{\mathrm{lb}}\beta n$ or the end of the path is reached. □

Hitting the Path

All that is left to complete the proof of the main result is the fact that the path is reached from a random initialization, with overwhelming probability.

With overwhelming probability we start with $\mathcal{B}_x = \emptyset$ and at least $\eta_{\mathrm{ub}}\beta n$ 0-bits in the window B_1. We maintain at least $\eta_{\mathrm{lb}}\beta n$ 0-bits in B_1, while the algorithm is encouraged to turn the 0-bits outside of B_1 to 1 quickly. Once the number of 0-bits outside of B_1 has decreased to or below αn, the path has been reached. Following the argumentation on small overlaps in Section 6.3.2 and recognizing that B_1 only has a small overlap to sets B_j that are far further on the path, i. e., $j > \beta n$, we see that the 0-bits in B_1 ensure that the initial i^*_x-value, i. e., the initial position on the path, is at most βn. Hence the (1+1) EA finds the start of the long path with overwhelming probability.

Lemma 6.21 (Doerr et al. (2011))**.** *Let $0 < \alpha < \beta < \gamma < 1$, $0 < \eta_{\mathrm{lb}} < \eta_{\mathrm{ub}} < 1/2$, and $c \ge 16$ be constants such that $\eta_{\mathrm{lb}} - \alpha/\beta > \gamma$ and $\beta/(1-\beta) < \gamma < 2\beta/(1-\beta)$, $c\beta > (2-2\eta_{\mathrm{ub}})/(1-2\eta_{\mathrm{ub}})$, and $\alpha < (1-2\eta_{\mathrm{ub}})/(3c)$. Let f_Π, with respect to α, β, and γ, be constructed as in Definition 6.12, for Π chosen uniformly at random. With probability $1 - 2^{-\Omega(n)}$ the (1+1) EA with mutation probability $p(n) = c/n$ optimizing f_Π at some point of time reaches some search point x with $i^*_x \le \beta n$ and $|x_{|B_{i^*_x}}|_0 \ge (\eta_{\mathrm{lb}} + \eta_{\mathrm{ub}})/2 \cdot \beta n$.*

Proof. The proof follows from reusing many previous arguments as the situation of the (1+1) EA moving towards the path is very similar to climbing up the path. The outline of the proof is as follows. We first show that a minimum number of 0-bits in B_1—the same minimum number as in the setting of climbing the path—prevents the (1+1) EA from taking shortcuts. We then show that the path is reached within the first n^2 iterations. Finally, we argue that, during these n^2 iterations, we keep a minimum number of 0-bits inside the window, with overwhelming probability.

Let x be the current search point of the (1+1) EA. By the same reasoning as in Lemma 6.16 we observe that if $|x_{|B_1}|_0 \ge \eta_{\mathrm{lb}}\beta n$ then for every $j > \beta n$, since $\eta_{\mathrm{lb}}\beta n > \alpha n + \gamma\beta n$ and $|B_1 \cap B_j| \le \gamma\beta n$, we have $j \notin \mathcal{B}_x$. Hence, we only need to prove that the number of 0-bits in B_1 does not decrease below $\eta_{\mathrm{lb}}\beta n$ until the set $\mathcal{B}_x$ of potential window positions becomes non-empty for the first time.

The set $\mathcal{B}_x$ is non-empty if the number of 0-bits outside of B_1 has decreased towards a value of at most αn. Every mutation decreasing the number of 0-bits outside of B_1 is accepted. Such a mutation has a probability of at least

$$\alpha n \cdot \frac{c}{n}\left(1 - \frac{c}{n}\right)^{n-1} = \Omega(1).$$

Hence, there is a constant $\kappa > 0$ such that for any initialization the expected number of iterations until the number of 0-bits has decreased to a value of at most αn is at most κn. By Markov's inequality (Lemma B.20), the probability that this has not happened after $2\kappa n$ iterations is at most $1/2$. The probability that this still has not happened after $\lfloor n^2/(2\kappa n)\rfloor = \Omega(n)$ periods, each of $2\kappa n$ iterations, is $2^{-\Omega(n)}$. Hence, with probability $1 - 2^{-\Omega(n)}$ the path is found within n^2 iterations. In the following we assume that this happens.

Recall that we have $|B_i| = \ell = \beta n$ and $\eta_{\mathrm{ub}} < 1/2$, constant. The initial search point contains an expected number of $1/2 \cdot \beta n$ 0-bits in B_1. The probability that the initial search point contains at least $\eta_{\mathrm{ub}}\beta n$ 0-bits in B_1 is $1 - 2^{-\Omega(n)}$ by Chernoff bounds (Lemma B.21). Assume that this happens and consider a situation where we have at least αn 0-bits outside of B_1 and the number of 0-bits in B_1 has decreased below $\eta_{\mathrm{ub}}\beta n$. Arguing as in the proof of Lemma 6.18, if the number of 0-bits in B_1 is within $\eta_{\mathrm{lb}}\beta n$ and $\eta_{\mathrm{ub}}\beta n$ then there is a positive drift towards increasing the number of 0-bits again. The only difference to the previous arguments is as follows. Instead of considering a new random BinVal instance when the current i^*_x-value is increased, we obtain a new BinVal instance whenever the number of 1-bits outside the window is increased. The probability for the latter event can even be larger than the probability for the former. This allows us to apply Lemma 6.19 in the same fashion as in the proof of Lemma 6.18. This results in a positive drift. Since we start with at least $\eta_{\mathrm{ub}}\beta n$ 0-bits in B_1, we can apply the drift theorem as in Theorem 6.20 w. r. t. the interval $[(\eta_{\mathrm{lb}}+2\eta_{\mathrm{ub}})/3\cdot\beta n, \eta_{\mathrm{ub}}\beta n]$. This proves that in n^2 iterations the number of 0-bits in B_1 does not drop to or below $(\eta_{\mathrm{lb}} + 2\eta_{\mathrm{ub}})/3 \cdot \beta n$, with probability $1 - 2^{-\Omega(n)}$.

We only have to deal with one further caveat. If x^* is the first point on the path then the above arguments on the 0-bits inside B_1 do not apply for the iteration in which x^* is created. This is because every point on the path is better than every point y with $\mathcal{B}_y = \emptyset$, and so selection works differently in this special iteration. We resort to a more direct argument to prove that not many 0-bits are lost. Consider the mutation that creates x^*. Since x^* is the first search point where $\mathcal{B}_{x^*} \neq \emptyset$, its parent must have had more than αn 0-bits outside of B_1. It also had at least $(\eta_{\mathrm{lb}} + 2\eta_{\mathrm{ub}})/3 \cdot \beta n$ 0-bits inside B_1. The probability that during mutation more than $(\eta_{\mathrm{ub}} - \eta_{\mathrm{lb}})/6\cdot\beta n$ bits were flipped is $n^{-\Omega(n)}$. Hence with overwhelming probability the number of 0-bits in x^* is still at least

$$\begin{aligned} |x^*|_0 &\geq \alpha n + \frac{\eta_{\mathrm{lb}} + 2\eta_{\mathrm{ub}}}{3} \cdot \beta n - \frac{\eta_{\mathrm{ub}} - \eta_{\mathrm{lb}}}{6} \cdot \beta n \\ &\geq \alpha n + \frac{\eta_{\mathrm{lb}} + \eta_{\mathrm{ub}}}{2} \cdot \beta n. \end{aligned}$$

Along with Lemma 6.13, this yields that $|x^*_{|B_{i^*_x}}|_0 = |x|_0 - \alpha n \geq (\eta_{\mathrm{lb}} + \eta_{\mathrm{ub}})/2 \cdot \beta n$ as claimed. As the sum of all error probabilities is $2^{-\Omega(n)}$, the claim follows. □

Putting Everything Together

We are now prepared to prove Theorem 6.15.

Proof of Theorem 6.15. Choose $\eta_{\text{lb}} := 30/112$ and $\eta_{\text{ub}} := 30/111$. It is easily verified that for the chosen values $0 < \alpha < \beta < \gamma < 1$, $c\beta > (2-2\eta_{\text{ub}})/(1-2\eta_{\text{ub}})$, $\eta_{\text{lb}} - \alpha/\beta > \gamma$, $\beta/(1-\beta) < \gamma < 2\beta/(1-\beta)$, and $\alpha < (1-2\eta_{\text{ub}})/(3c)$ holds, satisfying all preconditions on these variables for Lemmas 6.16, 6.20, and 6.21. By Lemma 6.21 the (1+1) EA reaches some search point x with $i_x^* \le \beta n$ and $|x_{|B_{i_x^*}}|_0 \ge (\eta_{\text{lb}} + \eta_{\text{ub}})/2 \cdot \beta n$ with overwhelming probability. Lemma 6.20 then states that with probability $1-2^{-\Omega(n)}$ the number of 0-bits in the current window is always at least $\eta_{\text{lb}}\beta n$ until the end of the path is reached or $2^{\kappa n}$ iterations have passed for a sufficiently small constant $\kappa > 0$ (which would correspond to the claimed time bound).

Given the condition on the 0-bits, by Lemma 6.16 the (1+1) EA increases its current i_x^*-value by at most βn in one iteration, with probability $1-n^{-\Omega(n)}$. The probability that this always happens until an i_x^*-value of L' is reached is at least $1-L' \cdot n^{-\Omega(n)} = 1-n^{-\Omega(n)}$ since $L' = 2^{\Theta(n)}$. This implies that (1+1) EA spends at least $L'/(\beta n) - 1 \ge 2^{\kappa n}$ iterations on the path, with probability $1 - 2^{-\Omega(n)}$, if κ is chosen small enough. Since the sum of all error probabilities is $2^{-\Omega(n)}$, the claim follows. □

In this chapter, we have seen that there exists a class of monotone functions where increasing the mutation probability from $1/n$ to c/n for a constant $c \ge 16$ increases the optimization time from polynomial to exponential. Since the definition of this function class is rather complicated, the question on the practical relevance of this result arises immediately. In the preceding chapters we performed experiments in order to answer this question. However, since we have only shown a negative result here, experiments are in general not too useful. Moreover recall, that $L := \lfloor \exp\bigl((\gamma-\rho)^2(1-\beta)n/6\bigr) \rfloor$ (Lemma 6.10). Plugging in the concrete values stated below eq. (6.1), yields that $L < n$ as long as $n < 500{,}000$. Thus, the definition makes only sense for very large values of n, making the design of appropriate experiments even harder if not impossible.

In addition, we did not describe a concrete construction procedure for the function class. Thus, it seems to be necessary to perform the random experiment described in order to obtain such a function. This involves the generation of an exponential number of windows, for which the condition on the amount of overlapping positions has to be checked. Afterwards, the optimization process can be started. Besides the enormous preprocessing time this procedure yields exponential space requirements. We conclude that experimental supplements do not make much sense in the context of this chapter. We remark that it is an interesting open problem to find out, if there are practical relevant or at least easier to evaluate (monotone) functions where increasing the mutation probability from $1/n$ to c/n for some not too large constant c increases the optimization time from a small polynomial to exponential with overwhelming probability.

Part III.

Analyses of Aging in Artificial Immune Systems

7. Introduction

There are many different ways of tweaking the performance of randomized search heuristics one being the addition of more advanced and sometimes rather complicated mechanisms. One such mechanism is aging where each point in the search space is equipped with an individual age and ages in each round of the search heuristics. A maximal lifespan $\tau_{\max}$ is introduced and each search point with an age exceeding $\tau_{\max}$ is removed from the current collection of search points making room for new and perhaps more promising search points. The mechanism of aging is thought of as increasing the diversity of the collection of search points and is hoped to be helpful for multi-modal problems where simpler search heuristics may get stuck in local optima.

Aging has been used in different kinds of randomized search heuristics and for various decisions made during the optimization process. Typical applications are

- in the selection for replacement, see e. g., Schwefel and Rudolph (1995), Kubota and Fukuda (1997), António (2006), Hornby (2006, 2009, 2010), Cutello et al. (2004a), Cutello and Nicosia (2002b), and many more,
- in the selection for reproduction, see e. g., Ghosh et al. (1997) or Mak and Lau (2008),
- for controlling the mutation strength, see e. g., (Kim et al. 1996), or
- for controlling the size of the collection of search points, see e. g., Michalewicz (1996), Bäck et al. (2000), or Cutello et al. (2006a).

Hence, it is not surprising that there exists a large variety of different aging operators. We present a brief overview of these approaches within the next section. Since, in the context of this thesis, we concentrate on the use of aging in the selection for replacement only, we restrict our considerations to these approaches. Afterwards an overview over results presented in this part of the thesis is given. We close the introductory part by presenting the aging-based algorithmic framework that is used in our analyses.

7.1. Related Work

In the area of randomized search heuristics aging is in particular used in evolutionary computation and in artificial immune systems. All approaches considered here have in common that an additional parameter $\tau_{\max}$ is introduced into the algorithm. Moreover, each search point is assigned an age which is increased by 1 in each iteration, or alternatively, when a search point was selected for reproduction. However, different types of aging differ in the implementation of the initialization of the age for new search points and the decision process whether a search point should be removed due to its age.

In evolutionary computation aging is mostly used by assigning age 0 to each new offspring and replacing search points exceeding the pre-defined maximal lifespan $\tau_{\max}$ deterministically, independently of their current function value (Schwefel and Rudolph 1995). We call this type of aging *evolutionary aging*. In artificial immune systems a different kind of aging, called *static pure aging*, is more common (Cutello et al. 2004a). Here, in contrast to the former version new search points inherit by default the age of their parent and are only assigned age 0 if their function value is strictly larger than that of their parents. This aging scheme intends to give an equal opportunity to each improving new search point to effectively explore the landscape. In the static pure aging variant search points exceeding the maximal lifespan $\tau_{\max}$ are again removed deterministically.

We remark that there exist approaches, where the initial age is set to a random value in $[0, c\tau_{\max}]$ for some constant $0 < c < 1$, e. g., Cutello et al. (2010). Additionally, there exists a *stochastic aging* variant (Cutello and Nicosia 2002b), where each search point is removed with probability $1 - 2^{-1/\tau_{\max}}$. In this context $\tau_{\max}$ is referred to as the expected mean life time (sic!), observed in biological processes (Seiden and Celada 1992). The actual expected life time is, however, approaching $\tau_{\max}/\ln 2$. We remark, that elitist versions of both aging variants exist (Cutello et al. 2007c). In order to keep the size of the collection of search points constant at a certain size μ, often new random search points with age 0 are introduced, if necessary.

First attempts to incorporate aging into a randomized search heuristic took place in the field of evolutionary computation. One of the first approaches was presented by Schwefel and Rudolph (1995) within the so-called $(\mu, \kappa, \lambda, \rho)$ evolution strategy (ES) for continuous optimization where $\kappa \geq 1$ corresponds to our notion of a maximal lifespan $\tau_{\max}$. This strategy combines the advantages of the well-known (μ, λ)-ES and the $(\mu+\lambda)$-ES by allowing more flexibility to choose between these two extreme cases. However, the two basic strategies can be easily obtained by setting $\kappa = 1$ (comma strategy) and $\kappa = \infty$ (plus strategy), respectively. Note, that for finite κ, we additionally need $\lambda \geq \mu$ in order to guarantee a constant size μ of the collection of search points during the whole optimization process.

Another noteworthy occurrence of aging in the field of evolutionary computation is the recently introduced age-layered population structure (ALPS) (Hornby 2006, 2009, 2010). This approach exhibits similarities to aging as used in artificial immune systems. It restricts competition between search points of different age within the collection of search points and thus, allows younger search points to improve without being replaced by older search points. In contrast to the evolutionary aging scheme described above, age in ALPS is not related to the number of iterations a search point has been part of the collection of search points but rather describes how long the 'genetic material' has been evolving. Thus, search points inherit the age of their oldest parent and both, the age of the parents and the offspring are increased by 1 at the end of an iteration. ALPS is considered to be rather successful in preserving a certain degree of diversity within the collection of search points. A multi-objective algorithm inspired by ALPS was presented by Schmidt and Lipson (2011).

In the field of artificial immune system most work about aging was executed within the well-known clonal selection algorithm opt-IA (Cutello et al. 2004a) and its relatives. To the best of our knowledge, the first occurrence of stochastic aging in the literature is in Cutello and Nicosia (2002b) while static pure aging was first mentioned by Cutello et al. (2004a,b). Up to now there are a number of different applications, but in most publications it remains unclear if aging is a crucial part in these algorithms. For example aging was used in the context of protein structure prediction (Cutello et al. 2007b), the chromatic number problem (Cutello et al. 2007a), graph coloring (Cutello et al. 2003), multiple sequence aligning (Cutello et al. 2006b), string and protein folding (Cutello et al. 2005a), and economic load dispatch (Gonçalves et al. 2007). An experimental analysis for static pure aging on dynamic optimization problems was carried out by Castrogiovanni et al. (2007). A convergence analysis for an algorithm incorporating aging was presented by Cutello et al. (2007c). Note, that this is not an exhaustive list of references. A complete overview is out of scope of this thesis.

7.2. Contribution of this Thesis

Up to now, the analysis of the aging mechanisms described above have been mostly empirical. Since aging is often part of rather specialized and advanced algorithms, it is not immediately clear what aging does and where the benefits of aging are. However, it is claimed that aging helps in preserving a certain degree of diversity (Cutello et al. 2007b). Thus, it is an interesting question to investigate what aging can achieve during an optimization process. We will do so in this part of the thesis and contribute to the theoretical foundations of such operators.

We apply aging during the selection for replacement process of a randomized search heuristic and concentrate on the deterministic, non-elitist variants described above, i. e., evolutionary aging and static pure aging. As we are in particular interested in the effects of aging, we consider these operators in isolation within an algorithm framework similar to the one described in the previous part of this thesis. We describe an extension of this framework in the next section. Note, that we present results of experiments in each chapter of this part in order to investigate the practical relevance of our rigorous results.

As in the part about mutation operators, we start with an analysis of the most important parameter of aging, namely the maximal lifespan $\tau_{\max}$ (Chapter 8). In applications this parameter is often set to some 'magic' value derived from preliminary experiments. In many cases this value seems to be independent of the size of the search space which is somehow a similar observation to the parameter ρ from inversely fitness-proportional mutation probabilities discussed in Chapter 4. Therefore, we analyze different settings of the maximal lifespan $\tau_{\max}$ and derive guidelines for setting this parameter in practical applications.

Based on these results we compare different aging operators in Chapter 9. On one hand we consider evolutionary aging known from the field of evolutionary computation.

On the other hand, we investigate static pure aging from the field of artificial immune systems. We show on which kind of problems these two operators have their strengths and weaknesses. Finally, we introduce a third operator that provably shares the advantages of the two former aging mechanisms.

In the last chapter of this part (Chapter 10), we establish a broader view on aging, in particular with respect to the interplay of aging with the selection for replacement. We investigate the role of age diversity and how such diversity can be achieved using different selection for replacement strategies. Based on the observation, that in the two previous chapters static pure aging mainly executed some kind of restart strategy, our main attention in this chapter is the question what aging can achieve beyond such restarts.

7.3. An Extended Algorithmic Framework

We aim at analyzing the effects of different aging operators on the performance of randomized search heuristics. Similar to the analyses of different mutation operators in Part II, we study this by implementing the different kinds of aging in a minimal algorithmic framework. In order to keep things as simple as possible and allow for easier comparisons, we extend the framework from Section 3.3 with the necessary aging modules. We point out important similarities and differences of these two frameworks.

The randomized search heuristic uses again a collection of search points of size μ. It works in rounds where in each round all search points grow older, one new search point is generated as random variation of existing search points, its age is decided, search points that are too old are removed and new randomly generated search points are introduced to keep the number of search points constant at μ. A more formal description of the algorithmic framework is given in Algorithm 7.1.

Algorithm 7.1 An Extended Algorithmic Framework.

```
1. Initialization
   Let $t := 0$ and initialize collection of search points $C_0$ of size $\mu$.
2. Repeat
   a) Let $t := t + 1$.
   b) Aging: Growing older
      Increase age of all search points in $C_{t-1}$.
   c) Variation
      Generate new search point $y$.
   d) Aging: Age of new search points
      Decide about the age of the new search point $y$.
   e) Aging: Removal due to age
      Remove search points with age exceeding $\tau_{\max}$.
   f) Selection for Replacement
      Decide if $y$ is to be inserted in $C_t$. Remove or add search points as needed.
   Until some termination condition is met.
```

As in Part II, we use Algorithm 7.1 for maximization of a function $f\colon \{0,1\}^n \to \mathbb{R}$ and implement the (in this case) seven modules in very simple ways. The initialization (line 1[1], Algorithm 7.2) is again carried out uniformly at random. Due to the use of aging, all search points are additionally assigned age 0.

Algorithm 7.2 Initialization.

```
1. Choose $x_1, \dots, x_\mu \in \{0,1\}^n$ uniformly at random.
2. **For all** $i \in \{1, \dots, \mu\}$
3.     Set $x_i$.age := 0.
4. Set $C_0 := \{x_1, \dots, x_\mu\}$.
```

The search points grow older by 1 in each iteration of the main loop (line 2b, Algorithm 7.3). Variation creates one new search point y by means mutation of a search point selected uniformly at random from the current collection of search points (line 2c, Algorithm 3.3). In a first step we simply use standard bit mutation as introduced in Algorithm 3.5, i. e., each bit of the search point is flipped with probability $p(n)$. We choose $p(n) = 1/n$ here.

Algorithm 7.3 Aging: Growing Older

```
1. **For all** $x \in C_{t-1}$
2.     Set $x$.age := $x$.age + 1.
```

We analyze the two different aging operators described above, namely static pure aging and evolutionary aging. We give precise formal definitions of both operators as they become part our algorithmic framework (line 2d, Algorithm 7.4 and 7.5).

Algorithm 7.4 Static Pure Aging (spa).

```
1. **If** $f(y) > f(x)$ **then**
2.     Set $y$.age := 0.
3. **Else**
4.     Set $y$.age := $x$.age.
```

Following the common practice of aging operators all search points exceeding the maximal lifespan $\tau_{\max}$ are removed, i. e., not transfered to the next collection of search points C_t (line 2e, Algorithm 7.6).

In the selection for replacement (line 2f, Algorithm 7.7) we decide if the new search point is inserted into the current collection of search points or not. If at least one current search point is removed due to its age the new search point is inserted. Otherwise it is only inserted if its function value is not worse than the worst function value of any of the current search points. In this case it replaces one current search point that is selected uniformly at random among all search points with worst function value. In order to keep the size of the collection of search points constant at μ, we additionally fill up the

[1]all line numbers from Algorithm 7.1

Algorithm 7.5 Evolutionary Aging (eva).

```
1. Set $y$.age := 0.
```

Algorithm 7.6 Aging: Removal Due to Age.

```
1. Set $C_t := \{x \mid x \in C_{t-1}$ and $x$.age $\leq \tau_{\max}\}$
```

collection of search points by generating new search points uniformly at random and assigning them age 0.

Algorithm 7.7 Selection for Replacement.

```
1.  If $|C_t| < \mu$ then
2.      If $y$.age $\leq \tau_{\max}$ then
3.          Set $C_t := C_t \cup \{y\}$.
4.      While $|C_t| < \mu$ do
5.          Select $x \in \{0,1\}^n$ uniformly at random.
6.          Set $x$.age := 0. Set $C_t := C_t \cup \{x\}$.
7.  Else
8.      Choose $z \in C_t$ with minimal fitness uniformly at random.
9.      If $f(y) \geq f(z)$ then
10.         Set $C_t := (C_t \cup \{y\}) \setminus \{z\}$.
```

We denote Algorithm 7.1 using static pure aging from Algorithm 7.4 by A_{spa} and Algorithm 7.1 using evolutionary aging from Algorithm 7.4 by A_{eva}. Hence, again following the previous notation, we denote the optimization time of these algorithms on some function f by $T_{A_{\text{spa}},f}$ and $T_{A_{\text{eva}},f}$, respectively. Note, that setting the maximal lifespan $\tau_{\max} = \infty$ yields the well-known (μ+1) EA (see Definition 3.1).

We remark that our measure for the optimization time, the number of iterations, is smaller than the number of function evaluations, not only since we need μ evaluations for the initial collection of search points but also since we may need more than 1 evaluation per iteration since newly generated search points introduced due to aging (line 5-6 of Algorithm 7.7) also require function evaluations. Since each search point can be removed at most each $\tau_{\max}$-th iteration the number of function evaluations is bounded above by $\mu + (1 + (\mu/\tau_{\max})) \cdot T$, where T denotes the number of iterations executed.

8. Setting the Maximal Lifespan

In a first step, we consider the most important parameter of an aging operator, namely the maximal lifespan $\tau_{\max}$. We demonstrate that the choice of this parameter is both, crucial for the performance and difficult to set appropriately since aging is very sensitive with respect to $\tau_{\max}$. We investigate this degree of sensitivity by identifying families of functions for which there exists a phase transition from polynomial to exponential optimization times as $\tau_{\max}$ increases or decreases or if $\tau_{\max}$ lies outside a given interval that can be arbitrarily shifted and made arbitrarily small. Along the way, we present a general technique that can be used to strengthen the properties of an example function or to combine the properties of different example functions.

In the following, we mainly consider static pure aging since this is the operator most often used in artificial immune systems. We start with the derivation of a lower bound for the maximal lifespan in Section 8.1. Afterwards, we investigate in which situations a too large maximal lifespan can be harmful (Section 8.2). Using the method mentioned above, we then construct a function where an appropriate age lies within a very narrow range (Section 8.3). We close the chapter by presenting experiments complementing our theoretical analyses in Section 8.4. This chapter is mainly based on the results from Horoba et al. (2009). However, the results in Section 8.1.1, 8.1.4, and 8.4 were not published before.

8.1. Lower Bounds for the Maximal Lifespan in Static Pure Aging

Fitness functions can be of very different difficulty for randomized search heuristics. This is not only true with respect to optimization but also with respect to local progress measures. If static pure aging is employed, an important local measure is the time needed to create an offspring with strictly larger function value than its parent. Only in this case the offspring's age is set to 0, otherwise it inherits its parent's age. It is intuitively clear that if the maximal lifespan $\tau_{\max}$ is smaller than the time needed for such a local improvement the randomized search heuristic with aging is in trouble. It does not have sufficient time to create a better offspring. Parents are replaced by new random search points in this time. One may be tempted to believe that, if this happens anywhere during the course of each run with probability close to 1, a randomized search heuristic employing aging with a too small maximal lifespan $\tau_{\max}$ is lost. But things are less simple as we will see in the following.

We start our investigations by considering the simple example functions OneMax (Definition 4.1) and LeadingOnes (Definition 5.8). In Section 8.1.1 we derive lower bounds on the maximal lifespan for these two functions that will be used later when considering more complicated scenarios. Afterwards, we point out a situation in which the disadvantage of a too small maximal lifespan $\tau_{\max}$ can be compensated by means of a kind of 'restart' (Section 8.1.2). In Section 8.1.3 we present an parameterized function and a suitable lower bound on the maximal lifespan in this case. We close with some remarks on evolutionary aging in Section 8.1.4.

8.1.1. Some Simple Examples

As before, we start our considerations with some simple example functions in order to get an intuition about the processes involved in our algorithm. We start with an analysis of LeadingOnes and show how the parameter $\tau_{\max}$ influences the expected optimization time. In particular, we compare the resulting optimization time with that of the (μ+1) EA (Witt 2006). Recall, that the (μ+1) EA equals Algorithm 7.1 with $\tau_{\max} = \infty$. We remark that some of the ideas in the following proofs are from the proofs in Horoba et al. (2009). Thus, we can reuse the results when considering the parametrized example function from Horoba et al. (2009) in Section 8.1.3.

Theorem 8.1. *Let $\mu = n^{O(1)}$, $c > 6e$, and $\tau_{\max} \geq c \log n \cdot (n + \mu \log n)$. The expected optimization times of A_{spa} and the* (μ+1) EA *on* LeadingOnes *are*

$$E\bigl(T_{(\mu+1)\text{ EA, LeadingOnes}}\bigr) \leq \mu + 3en \cdot \max\{\mu \ln(en), n\} = O\bigl(n^2 + \mu n \log n\bigr)$$

$$E\bigl(T_{A_{spa},\text{LeadingOnes}}\bigr) \leq \frac{n^{\frac{c}{6e}-1}}{n^{\frac{c}{6e}-1} - 1} \cdot E\bigl(T_{(\mu+1)\text{ EA, LeadingOnes}}\bigr) = O\bigl(n^2 + \mu n \log n\bigr).$$

Proof. The result for the (μ+1) EA was already proven by Witt (2006; Theorem 1). Thus, we are only left with the second statement. We revisit the proof by Witt (2006) and adapt it to the aging-based variant A_{spa}.

The first crucial observation is that we need not worry about the complete optimization time of the algorithm on LeadingOnes. Each search point that is an improvement in comparison to its parent starts with age 0. Thus, it suffices if $\tau_{\max}$ is sufficiently large in comparison to the time needed to increase the current best function value once. In order to do so, it suffices to select a currently best search point and to flip only the leftmost 0. The probability for the latter event is $(1/n) \cdot (1 - 1/n)^{n-1}$. Thus, the probability to increase the currently best function value is at least

$$\frac{i}{\mu} \cdot \frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{i}{e\mu n},$$

where i is the number of best search points. The expected waiting time for this event is $e\mu n/i$. Moreover, the probability to make a copy of a current best increasing their number by 1 is $(i/\mu) \cdot (1 - 1/n)^n \geq i/(2e\mu)$. We see that the expected time needed to increase their number from at least 1 to at least b is bounded by

$$\sum_{i=1}^{b-1} \frac{2e\mu}{i} \leq 2e\mu(\ln b + 1) = 2e\mu \ln(eb) = O(\mu \log b) = O(\mu \log \mu)$$

where the two last equalities hold since $b \leq \mu$ and μ is polynomially bounded. Analogously to Witt (2006), we pessimistically assume that the best function value does not increase until at least $\min\{n/\ln(en), \mu\}$ best search points are in the collection of search points. Note, that $\ln(en) \leq \log(n)$ for $n \geq 4$. This yields

$$\begin{aligned}
2e\mu \ln\left(\frac{en}{\ln(en)}\right) + \frac{e\mu n}{\min\left\{\frac{n}{\ln(en)}, \mu\right\}} &\leq 2e\mu\ln(en) + \max\{e\mu\ln(en), en\} \\
&\leq 3e \cdot \max\{\mu\ln(en), n\} \\
&= 3e \cdot \max\{\mu(\ln n + 1), n\} \\
&\leq 3e(\mu \log n + n) \qquad (8.1) \\
&= O(n + \mu \log n)
\end{aligned}$$

as upper bound on the expected waiting time for increasing the current best function value. Therefore, one may be tempted to believe that $\tau_{\max} = O(n + \mu \log n)$ suffices. This, however, is not correct. We need a sequence of $\Theta(n)$ of such improvements without ever failing. If we fail to make one such improvement in $\tau_{\max}$ iterations the current best search point may be removed due to its age and we have to start again. Thus, we need the maximal lifespan $\tau_{\max}$ to be sufficiently large to make even a single failure in a sequence of $\Theta(n)$ needed improvements sufficiently unlikely.

Remember, that the expected time needed for an improvement is bounded above by $3e(n + \mu \log n)$ (eq. (8.1)). The probability not to have such an event in $6e(n + \mu \log n)$ steps is bounded above by $1/2$ (Markov inequality, Lemma B.20). We may consider $c \log n \cdot (n + \mu \log n)$ steps as $c \log n$ rounds of $(n + \mu \log n)$ steps each, or analogously $c \log n/(6e)$ rounds of $6e(n+\mu \log n)$ steps. For each round we have success probability at least $1/2$. Thus, the probability not to see any success in $c \log n/(6e)$ rounds is bounded above by $(1/2)^{c \log n/(6e)} = 1/n^{c/(6e)}$. We see that with $\tau_{\max} = c \cdot \log n\,(n + \mu \log n)$ the probability to increase the best function value is bounded below by $1 - n^{-c/(6e)}$. Using the simple union bound (Lemma B.10) we see that with probability at least

$$1 - n \cdot n^{-\frac{c}{6e}} = 1 - n^{1-\frac{c}{6e}} = 1 - o(1) \qquad (8.2)$$

we obtain the required sequence of at most n such improvements without ever failing.

In case of a failure, we are not in a worse situation than in the very beginning and we can pessimistically assume that a failure is equivalent to a complete restart of the algorithm. Due to the above calculation, we expect at most $O(1)$ restarts. This yields the claimed upper bound on the optimization time for A_{spa}. □

In the above theorem, it suffices to have a sequence of improvements without high probability. However, in the following we often need that this holds with higher probability. Setting $c = \omega(1)$, we immediately get the following desired corollary.

Corollary 8.2. *Let $\mu = n^{O(1)}$ and $\tau_{\max} = \omega(\log n\,(n + \mu \log n))$. The probability that A_{spa} succeeds in optimizing* LeadingOnes *without a 'restart' is $1 - n^{-\omega(1)}$.*

Proof. We only need to consider eq. (8.1) and observe that $1 - n^{1-\frac{c}{6e}} = 1 - n^{-\omega(1)}$ holds for $c = \omega(1)$. $\square$

We have seen which size of $\tau_{\max}$ is sufficient to yield the same asymptotical optimization time on LeadingOnes. However, decreasing the maximal lifespan below this bound may still lead to polynomial optimization times. Thus, we investigate for which values of $\tau_{\max}$ the considered algorithm A_{spa} only yields exponential optimization time with high probability. We demonstrate that A_{spa} needs to have a sufficiently large maximal lifespan in order to be successful on LeadingOnes given that the size of the collection of search points μ is reasonably bounded. The upper bound in Theorem 8.1 is dominated by the term involving the size of the collection of search points as soon as $\mu = \omega(n/\log n)$ holds. Thus, we restrict ourselves to smaller collection of search points sizes. As before we present lower bounds on the probability of having exponential optimization time since such results are much more informative than proving a large lower bound on the expected optimization time.

Theorem 8.3. *Let $\mu = O(n/\log n)$ and $\tau_{\max} = o(n \log n)$. The optimization time of A_{spa} on* LeadingOnes *is*

$$T_{A_{spa},\text{LeadingOnes}} = 2^{\Omega(n^{\varepsilon})}$$

($0 < \varepsilon < 1$ some arbitrary constant) with probability $1 - 2^{-\Omega\left(n^{1-o(1)}\right)}$.

Proof. Initially, we have LeadingOnes$(x) \leq n/2$ for all $x \in C_0$ with probability $1 - 2^{-\Omega(n)}$ (applying Chernoff bounds for a single search point and the union bound for the collection of search points). Then only mutations increasing the number of leading 1-bits are accepted. For each search point in the collection of search points, the bits right of the leftmost 0-bit are uniformly distributed. From this, it follows that $\Omega(n)$ improvements of the current best are needed (Droste et al. 2002).

Consider $2^{\Omega(n^{\varepsilon})}$ steps. Since for an improvement the leftmost 0-bit has to flip, the probability of increasing the current best function value is bounded above by $1/n$. The probability for such a mutation in r independent trials is bounded above by $1-(1-1/n)^r$. Remember that for the maximal lifespan $\tau_{\max} = o(n \log n)$ holds. Moreover, remember that the expected number of steps until the complete collection of search points consists of copies of the current best (provided that the current best does not improve in this time) is $O(\mu \log n)$. With $\mu = O(n/\log n)$ we have that the number of steps before the collection of search points gets extinct due to its age before reaching the optimum is $\tau_{\max} = o(n \log n)$. Thus, the number of trials to create an improved new search point is $r = \alpha(n)(n-1)\ln n$ for some α with $\lim\limits_{n\to\infty} \alpha(n) = 0$. Hence, each of the $\Theta(n)$ needed improvements is achieved with probability at most

$$1 - \left(1 - \frac{1}{n}\right)^{\alpha(n)(n-1)\ln n} \leq 1 - \frac{1}{n^{\alpha(n)}}$$

before the collection of search points is set back due to its old age. The probability that this happens at all $\Theta(n)$ improvements is bounded above by

$$\left(1 - \frac{1}{n^{\alpha(n)}}\right)^{\Theta(n)} \overset{(B.8)}{=} e^{-\Theta\left(n^{1-\alpha(n)}\right)} = 2^{-\Theta\left(n^{1-o(1)}\right)}.$$

The theorem follows by application of the union bound to the number of steps $2^{\Omega(n^{\varepsilon})}$. □

We have investigated bounds on the maximal lifespan $\tau_{\max}$ that lead to polynomial or exponential optimization time on LeadingOnes. In the remainder of this section, we derive similar results for OneMax.

Theorem 8.4. *Let $\mu = n^{O(1)}$, $c > 4e$, $\delta > 0$ constant and arbitarily small, and $\tau_{\max} \geq c \cdot (\mu \log \mu + n)$. The expected optimization times of A_{spa} and the $(\mu{+}1)$ EA on OneMax are*

$$E\left(T_{(\mu+1)\text{ EA, OneMax}}\right) \leq \mu + 5e\mu n + en\ln(en) = O(\mu n + n\log n)$$
$$E\left(T_{A_{spa},\text{OneMax}}\right) \leq \frac{e^{\frac{c}{e}} - 1}{e^{\frac{c}{e}} - 2 - \delta} \cdot E\left(T_{(\mu+1)\text{ EA, OneMax}}\right) = O(\mu n + n\log n)$$

Proof. The result for the $(\mu{+}1)$ EA was already proven by Witt (2006; Theorem 2). Thus, we are once again only left with the statement for A_{spa}. The proof follows the line of thought of Theorem 8.1 and uses ideas from Witt's proof.

Recall that we already discussed these proof ideas in Theorem 4.11 when considering inversely fitness-proportional mutation probabilities. There, it was shown that the probability to increase the current best function value is at least $(iz)/(e\mu n)$, where i is the number of best search points with z 0-bits each. The expected waiting time for this event is at most $e\mu n/(iz)$. Note, that this is also similar to the corresponding expression for LeadingOnes, but with the difference that the expected time for an improvement depends on the function value.

In contrast to Witt (2006), we pessimistically assume that the best function value does not increase until the whole collection of search points consists of best search points. Once, the collection of search points only contains best search points, the probability to increase the best function value is at least z/en. Thus, the probability not to see an improvement in cn trials can be bounded by

$$\left(1 - \frac{z}{en}\right)^{cn} \leq \left(\frac{1}{e}\right)^{\frac{cz}{e}}.$$

As already seen in Theorem 8.1, the expected time to increase the number of best search points from 1 to μ can be bounded above by $2e\mu\ln(e\mu)$. Note, that this process is equivalent to the coupon collector problem (Lemma B.17). Thus, the probability to need more than $\beta 2e\mu\ln(e\mu)$ ($\beta > 1$, constant) trials in order to copy the whole collection of search points can be bounded by $\mu^{-(\beta-1)}$ due to the coupon collector theorem. We apply the theorem for a large collection of search points of size $\mu \geq n^{1-\varepsilon}$ for some sufficiently small constant $0 < \varepsilon < 1$:

$$\operatorname{Prob}\left(T > \beta 2e\mu \ln(e\mu)\right) < \left(\frac{1}{\mu}\right)^{\beta-1} \leq \left(\frac{1}{n^{1-\varepsilon}}\right)^{\beta-1} = \frac{1}{n^{(1-\varepsilon)(\beta-1)}}$$

For smaller sizes of the collection of search points, i. e., $\mu < n^{1-\varepsilon}$, we need to be more careful. Note that in this case

$$\mu \log \mu < n^{1-\varepsilon} \log\left(n^{1-\varepsilon}\right) = (1-\varepsilon) n^{1-\varepsilon} \log n$$

holds and thus, $\tau_{\max} = c \cdot (\mu \log \mu + n)$ is dominated by the term cn. We consider $c'n$ steps as $c'n^{\varepsilon}/(4e(1-\varepsilon)\log n)$ rounds of $4e(1-\varepsilon)n^{1-\varepsilon}\log n$ steps. For each round we have success probability at least $1/2$ (Markov inequality). Thus, the probability not to see any success in $c'n^{\varepsilon}/(4e(1-\varepsilon)\log n)$ rounds is bounded above by

$$\left(\frac{1}{2}\right)^{\frac{c'}{4e(1-\varepsilon)} \cdot \frac{n^{\varepsilon}}{\log n}} = \left(\frac{1}{n}\right)^{\frac{c'}{4e(1-\varepsilon)} \cdot \frac{n^{\varepsilon}}{\log^2 n}}.$$

Note, that this is $o\big(1/n^{(1-\varepsilon)(\beta-1)}\big)$ if $(1-\varepsilon)(\beta-1) > 0$.

As for LeadingOnes, we need at most n improvements of the best function value, i. e., for $z \in \{n-1, n-2, \ldots, 1\}$. We use the simple union bound (Lemma B.10) to add up the failure probabilities for the different values of z. Note, that for each value of z there are two sources for a failure: on one hand the successful mutation (probability at most $e^{-cz/e}$) and on the other hand the copying of the collection of search points (at most $1/n^{(1-\varepsilon)(\beta-1)}$).

We add up the derived upper bounds of the failure probabilities for all steps. Note that, we need $(1-\varepsilon)(\beta-1) > 1$, leading to the stronger condition $\beta > 2$ in comparison to the coupon collector theorem. Moreover, if $\beta = 2 + \gamma$ for some constant $\gamma > 0$, we have $\varepsilon < \gamma/(1+\gamma)$ as a condition on ε. For a sufficiently small constant $\delta > 0$ we derive the following upper bound on the total failure probability.

$$\begin{aligned}
\sum_{j=1}^{n-1}\left(\frac{1}{n^{(1-\varepsilon)(\beta-1)}} + \left(\frac{1}{e^{\frac{c}{e}}}\right)^{j}\right) &= n \cdot \frac{1}{n^{(1-\varepsilon)(\beta-1)}} + \sum_{j=1}^{n-1}\left(\frac{1}{e^{\frac{c}{e}}}\right)^{j} \\
= n^{1-(1-\varepsilon)(\beta-1)} + \frac{\frac{1}{e^{\frac{c}{e}}} - \left(\frac{1}{e^{\frac{c}{e}}}\right)^{n}}{1 - \frac{1}{e^{\frac{c}{e}}}} &\leq n^{1-(1-\varepsilon)(\beta-1)} + \frac{\frac{1}{e^{\frac{c}{e}}}}{1 - \frac{1}{e^{\frac{c}{e}}}} \\
= n^{1-(1-\varepsilon)(\beta-1)} + \frac{1}{e^{\frac{c}{e}} - 1} \leq \frac{1+\delta}{e^{\frac{c}{e}} - 1}
\end{aligned}$$

We still need to discuss the lower bound on the constant c. Recall that $\tau_{\max} = c \cdot (\mu \log \mu + n)$ holds. In order to have $1/(e^{c/e} - 1) < 1$, we need $c > e \ln 2$, i. e., more than $(e \ln 2) \cdot n$ steps are needed in order to yield the desired probability for the n subsequent improvements. For $\mu \geq n^{1-\varepsilon}$, we consider $2e\beta\mu \log \mu > 4e\mu \log \mu$ steps for the coupon collector arguments. Thus, altogether we require $c > 4e$ in that case.

For $\mu < n^{1-\varepsilon}$, we consider $c'n$ steps for the copying process. Since $(e \ln 2) \cdot n$ steps are needed for the improvements , we still have $c' = c - e \ln 2$ left. With $c > 4e$ this yields $c' > 3e$. This suffices for the above argumentation. Altogether, this yields $c > 4e$.

Just like for LeadingOnes, we are, in case of a failure, not in a worse situation than in the very beginning and we pessimistically assume that a failure is equivalent to a complete restart of the algorithm. However, since for sufficiently large n the success probability is at least

$$1 - \frac{1+\delta}{e^{\frac{c}{e}} - 1} = \frac{(e^{\frac{c}{e}} - 1) - (1+\delta)}{e^{\frac{c}{e}} - 1} = \frac{e^{\frac{c}{e}} - 2 - \delta}{e^{\frac{c}{e}} - 1} \tag{8.3}$$

and we expect at most $(e^{c/e} - 1)/(e^{c/e} - 2 - \delta) - 1$ such restarts. This yields the claimed upper bound on the optimization time for A_{spa}. □

Similarly to Corollary 8.2, we show a corresponding result that holds with high probability and can be re-used later.

Corollary 8.5. *Let $\mu = n^{O(1)}$ and $\tau_{\max} = \omega(\mu \log \mu + n)$. The probability that A_{spa} succeeds in optimizing* OneMax *without a 'restart' is $1 - n^{-\omega(1)}$.*

Proof. We only need to consider eq. (8.3) and observe that $1-(1+\delta)/(e^{c/e-1)} = 1-n^{-\omega(1)}$ holds for $c = \omega(1)$. □

As for LeadingOnes we demonstrate that A_{spa} needs to have a sufficiently large maximal lifespan in order to be successful on OneMax.

Theorem 8.6. *Let $\mu = 1$ and $\tau_{\max} = O\big(n^{\delta}\big)$ for some constant δ with $0 < \delta < 1$. The optimization time of A_{spa} on* OneMax *is*

$$T_{A_{spa},\text{OneMax}} = 2^{\Omega(n^{\alpha})}$$

with probability $1 - 2^{\Omega(n^{\alpha})}$, $\alpha := \min\{\delta/6, (1-\delta)/6\}$.

Proof. Let $\tau_{\max} \le dn^{\delta}$ for some constant $d > 0$. Due to Chernoff bounds (Lemma B.21), the number of 0-bits is at least $n/3$ for all initial search points with probability $1-2^{-\Omega(n)}$.

Let z be the number of 0-bits in the current best search point. Consider the first point in time t where $z \le n^{1-\delta}$ holds. Let $\beta := \min\{\delta/3, (1-\delta)/3\}$. Similar to the application of the Chernoff bound in the proof of Theorem 4.15 we know that in one mutation at most n^{β} bits flip with probability $1 - n^{-\Omega(n^{\beta})}$. Thus, at time t we have $z > n^{1-\delta} - n^{\beta} > n^{1-\delta}/2$ with probability $1 - n^{-\Omega(n^{\beta})}$ for sufficiently large n.

The current best function value can only increase if at least one of the 0-bits is flipped. This event has probability z/n. Thus, with our bounds on z the probability to increase the current best function value within $\tau_{\max}$ steps is bounded above by

$$1 - \left(1 - \frac{z}{n}\right)^{\tau_{\max}} \overset{(B.8)}{\le} 1 - \left(\frac{1}{2e}\right)^{z\tau_{\max}/n} \le 1 - \left(\frac{1}{2e}\right)^{n^{1-\delta}dn^{\delta}/n} = 1 - \left(\frac{1}{2e}\right)^{d}.$$

Let $d' := (1/(2e))^d$. Note that $d' > 0$ is a constant.

We consider t iterations. The expected number of times the current best function value is increased in these $t = n^{O(1)}$ iterations is bounded above by tz/n. Due to Chernoff bounds (Lemma B.21) the probability to have at least $2tz/n$ such improvements is $2^{-\Omega(tz/n)}$. In $2tz/n$ improvements the number of 0-bits is in total decreased by at most $2t(z/n)n^\beta$ with probability $1 - tn^{-\Omega(n^\beta)} = 1 - n^{-\Omega(n^\beta)}$. We set $t := n^\beta \tau_{\max}$. With this choice after t iterations the number of 0-bits in the current best is bounded below by

$$\frac{n^{1-\delta}}{2} - 2n^\beta \tau_{\max} \cdot \frac{n^{1-\delta}}{n} \cdot n^\beta \geq \frac{n^{1-\delta}}{2} - 2cn^{\beta+\delta+(1-\delta)-1+\beta} \geq \frac{n^{1-\delta}}{3}$$

for sufficiently large n with probability $1 - n^{-\Omega(n^\beta)}$. Thus, the optimum has not yet been reached in these steps.

Since $t = n^\beta \tau_{\max}$ we have n^β phases of $\tau_{\max}$ iterations in these steps. In each of them we do not have an improvement of the current best with probability at least d'. Thus, the probability to have at least one phase without an improvement of the current best is $1 - (1-d')^{n^\beta} = 1 - 2^{-\Omega(n^\beta)}$. In this phase the current best search point is removed since it exceeds the maximal lifespan $\tau_{\max}$. We need to see a run without this event in order to reach the optimum. Application of the union bound yields that this does not happen within $2^{n^{\beta/2}}$ iterations with probability $1 - 2^{-\Omega(n^{\beta/2})}$. Since we have $\beta/2 = \alpha$ this concludes the proof. □

8.1.2. Saving Time by Effort

In the preceding section we have derived bounds on the maximal lifespan $\tau_{\max}$ that lead either to the same asymptotical optimization time as the (μ+1) EA or exponential optimization time. In all cases, it was essential to have small enough failure probabilities for events that are essential for the optimization process, like e. g., a successful mutation. Such an event is expensive if its probability is small since then the expected waiting time is large. One might come to the conclusion that it suffices to only consider the most expensive events and choose $\tau_{\max}$ large enough for those. However, this is not completely true since the crucial point in the proofs was the fact that n such events need to be possible without ever failing in between. If there is only a small number of expensive events, some kind of 'restart' might help optimizing the function efficiently.

For the sake of simplicity, we restrict ourselves to a single search point to make our point. Consider a function that is otherwise easy to optimize but where the search point is faced with a 'gap' where k specific bits need to be mutated simultaneously for some integer constant $k \gg 1$. The probability for such a mutation equals $(1/n)^k \cdot (1-1/n)^{n-k} < 1/n^k$. The waiting time is geometrically distributed (Lemma B.14) and thus larger than n^k. If we have the maximal lifespan $\tau_{\max}$ rather small, say $\tau_{\max} = n$, it is very unlikely that a parent will survive sufficiently long to create an offspring that 'jumps' over this 'gap'. But this may not be a problem if the average time t needed to get into this situation is small. With probability $\Theta(1/n^k)$ a search point arriving at this 'gap' will successfully create an offspring that manages to make this 'jump', so the expected number of times

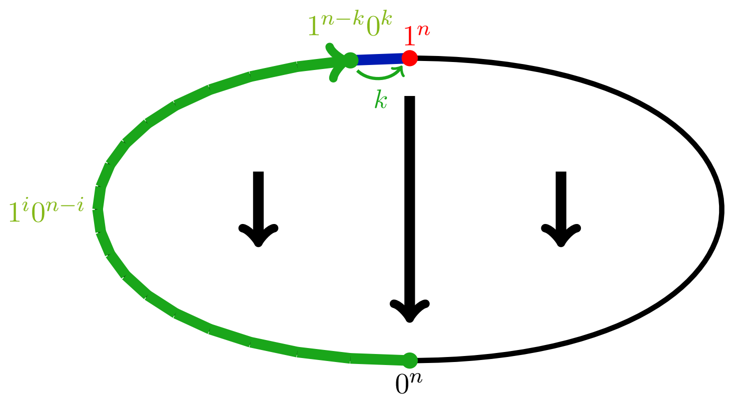


Figure 8.1.: The example function S-JUMP$_k$.

we need to have a parent at this 'gap' is $\Theta\big(n^k\big)$. Since we wait $\tau_{\max}$ iterations at the 'gap' before the search point is removed due to its age, we have $\tau_{\max}$ trials each time when we arrive at the gap. Thus, on average, after $O\big((t+\tau_{\max})\cdot n^k/\tau_{\max}\big) = O\big(t\cdot n^k\big)$ steps the algorithm can make it over the 'gap' even though the maximal lifespan is much too small for one such search point to wait for an improvement at the 'gap'. An example for such a function is depicted in Figure 8.1 and defined formally in Definition 8.7.

Definition 8.7. *For $n \in \mathbb{N}$ and $k \in \mathbb{N}$ with $k \geq 2$ the function* S-JUMP$_k : \{0,1\}^n \to \mathbb{N}$ *is defined by*

$$\text{S-JUMP}_k(x) := \begin{cases} 2n & \text{if } x = 1^n, \\ n+i & \text{if } x = 1^i0^{n-i}, i \in \{0,1,\dots,n-k\}, \\ n - \text{ONEMAX}(x) & \text{otherwise} \end{cases}$$

Theorem 8.8. *Let $\mu = 1$ and $\tau_{\max} = \omega(n \log n)$. The expected optimization times of A_{spa} and the* (1+1) EA *on* S-JUMP$_k$ *are*

$$E\left(T_{(\mu+1)\text{ EA, S-JUMP}_k}\right) = O\Big(n^k\Big)$$
$$E\left(T_{A_{spa},\text{S-JUMP}_k}\right) = O\left(\frac{n^{k+2}}{\tau_{\max}} + n^k\right)$$

Proof. The result for the (1+1) EA follows directly from the expected waiting time for the 'jump'.

Since $\tau_{\max} = \omega(n \log n)$, on average A_{spa} arrives at the gap after $O\big(n^2\big)$ iterations. The probability to find the optimum is then

$$1 - \left(1 - \left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k}\right)^{\tau_{\max}} \geq 1 - \left(1 - \frac{1}{en^k}\right)^{\tau_{\max}} \geq 1 - e^{-\tau_{\max}/\left(en^k\right)}.$$

For $\tau_{\max} \leq en^k$ this is bounded below by $\tau_{\max}/\left(2en^k\right)$ (Lemma B.9). Thus, in expectation $2en^k/\tau_{\max}$ restarts are sufficient. For $\tau_{\max} > en^k$ we have $1 - e^{-\tau_{\max}/\left(en^k\right)} = \Omega(1)$ and see that in expectation $O(1)$ restarts are sufficient. This implies an upper bound of $O\left(\left(n^2 + \tau_{\max}\right) \cdot \left(n^k/\tau_{\max}\right) + n^k\right)$. □

8.1.3. A Parameterized Example Function

We have seen that a class of example functions demonstrating that setting the maximal lifespan too small can be fatal needs to be slightly more involved than the one described in the previous section. We combine the well-known example function Ridge (Quick et al. 1998) (see Definition 8.9) with the kind of 'gap' we discussed before. The main difference is that we introduce a large number of such 'gaps' that all need to be 'jumped over' sequentially without a single failure in order to optimize this function (compare Figure 8.2). This can be proven to be exceedingly unlikely if the maximal lifespan $\tau_{\max}$ is too small. For a sufficiently large maximal lifespan $\tau_{\max}$, however, 'jumping over' those 'gaps' is rather simple. Note, that this is similar to the effects we have seen on OneMax and LeadingOnes. We define the concrete function considered in Definition 8.10.

Definition 8.9. *For $n \in \mathbb{N}$ the function* Ridge: $\{0,1\}^n \to \mathbb{N}$ *is defined by*

$$\textsc{Ridge}(x) := \begin{cases} n + i & \text{if } x = 1^i 0^{n-i}, \\ & i \in \{0, 1, \ldots, n\}, \\ n - \textsc{OneMax}(x) & \text{otherwise.} \end{cases}$$

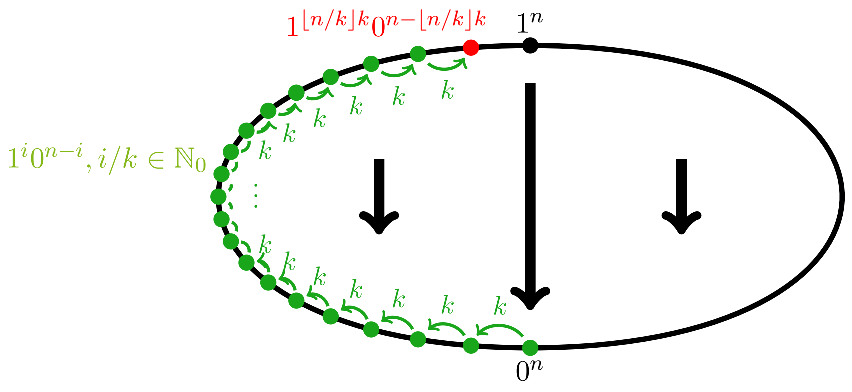
Definition 8.10. *For $n \in \mathbb{N}$ and $k \in \{1, 2, \ldots, n\}$ the function $f_k \colon \{0,1\}^n \to \mathbb{N}$ is defined by*

$$f_k(x) := \begin{cases} 0 & \text{if } x = 1^i 0^{n-i}, \\ & i \in \{0, 1, \ldots, n\}, \\ & \text{and } i/k \notin \mathbb{N}_0 \\ \textsc{Ridge}(x) & \text{otherwise.} \end{cases}$$

Note that $f_1 = $ Ridge. First we convince ourselves that f_k is not too difficult to optimize. We consider the (μ+1) EA as well as a A_{spa} where the maximal lifespan $\tau_{\max}$ is sufficiently large.

Theorem 8.11 (Horoba et al. (2009))**.** *Let $\mu = n^{O(1)}$, $\tau_{\max} = \omega\left(\log n \left(n^k + \mu \log n\right)\right)$, and $k = O(1)$. The expected optimization times of A_{spa} and the* (μ+1) EA *on f_k are*

$$E\left(T_{(\mu+1)\ \text{EA}, f_k}\right) = O\left(n^{k+1} + \mu n \log n\right)$$
$$E\left(T_{A_{spa}, f_k}\right) = O\left(n^{k+1} + \mu n \log n\right).$$



$1^{\lfloor n/k \rfloor k} 0^{n-\lfloor n/k \rfloor k}$
1^n
$1^i 0^{n-i}, i/k \in \mathbb{N}_0$
k
0^n

Figure 8.2.: The example function f_k.

Proof. We partition the search space $\{0,1\}^n$ into three disjoint sets that match the structure of f_k.

$$\begin{aligned} P_1 &:= \left\{1^i 0^{n-i} \mid i \in \{0,1,\dots,n\} \wedge i/k \notin \mathbb{N}_0\right\} \\ P_2 &:= \{0,1\}^n \setminus \left\{1^i 0^{n-i} \mid i \in \{0,1,\dots,n\}\right\} \\ P_3 &:= \left\{1^i 0^{n-i} \mid i \in \{0,1,\dots,n\} \wedge i/k \in \mathbb{N}_0\right\} \end{aligned}$$

Clearly, P_1, P_2, and P_3 form an f_k-based partition (Droste et al. 2002) with $P_1 <_{f_k} P_2 <_{f_k} P_3$. Let T_{P_1} denote the number of steps where we have $C_t \subseteq P_1$. Let T_{P_2} denote the number of steps where we have $C_t \subseteq P_1 \cup P_2$ and $C_t \cap P_2 \neq \emptyset$. Finally, let T_{P_3} denote the number of steps where we have $C_t \cap P_3 \neq \emptyset$ and $\max\{fk(x) \mid x \in C_t\} < \max\{fk(x) \mid x \in \{0,1\}^n\}$. Clearly, $T_{A,f_k} = T_{P_1} + T_{P_2} + T_{P_3}$ for $A \in \{A_{\text{spa}}, (\mu{+}1)\text{ EA}\}$ holds.

Since the most important part of f_k is in P_3 we begin with estimating $\mathrm{E}(T_{P_3})$. The other parts serve only the purpose of guiding the search heuristic to P_3. We follow the line of thought of Theorem 8.1.

In P_3, we need mutations of k specific bits in order to create an offspring with larger function value. This situation occurs $\Theta(n)$ times. Each time the probability for such a mutation increasing the current best function value is at least $(i/\mu) \cdot (1/n)^k \cdot (1-1/n)^{n-k} \geq (i/\mu) \cdot 1/(en^k)$, where i is the number of best search points. Moreover, the expected time to increase the number of best search points from at least 1 to at least b is bounded by $O(\mu \log b)$. Together this yields $O\big(n^k + \mu \log n\big)$ as upper bound on the expected waiting time for one member of the collection of search points 'jumping over' one such 'gap'. Since there are $\Theta(n)$ such 'gaps', we have $\mathrm{E}(T_{P_3}) = O\big(n^{k+1} + \mu n \log n\big)$ for the $(\mu{+}1)$ EA.

If the maximal lifespan $\tau_{\max}$ is $\tau_{\max} = \omega\big(n^{k+1} + \mu n \log n\big)$ this bound carries over to A_{spa} directly. However, since we only assume $\tau_{\max} = \omega\big(\log n\, (n^k + \mu \log n)\big)$ we need to be more careful. We can easily adopt the argumentation from Theorem 8.1 since again we need not worry about the complete time in T_{P_3}. Similarly to the proof of

Theorem 8.1, we see that with $\tau_{\max} = \omega\big(\log n \left(n^k + \mu \log n\right)\big)$ the probability to be successful at one such 'gap' is bounded below by $1 - n^{-\omega(1)}$ and by a simple union bound we have $\operatorname{Prob}\left(T_{P_3} \leq n \cdot \log(n) \cdot 2c(n^k + \mu \log n)\right) = 1 - n \cdot n^{-\omega(1)} = 1 - n^{-\omega(1)}$. Thus, if we only take into account the time spent in P_3 this yields $\mathrm{E}\left(T_{P_3}\right) = O\big(n^{k+1} + \mu n \log n\big)$ for A_{spa}.

Now we bound the expected time we need to get to P_3. First, we see that with probability $1 - 2^{-\Omega(n)}$ the initial collection of search points is disjoint to P_1. Moreover, for each $x \in P_1$ the probability that a mutation leads to some $y \notin P_1$ is $\Omega(1)$. Therefore, P_1 has negligible contribution to the expected optimization time, $\mathrm{E}\left(T_{P_1}\right) = O(1)$.

In P_2 the function values are given by n − OneMax. Due to symmetry reasons we may consider OneMax instead. Since it suffices to reach any $x \in P_3$ to leave P_2, any upper bound on the expected optimization on OneMax is an upper bound on T_{P_2}. Using Theorem 8.4 we get $\mathrm{E}\left(T_B\right) = O(\mu n + n \log n)$. Due to Corollary 8.5 this also holds with probability $1 - n^{-\omega(1)}$.

By adding up the expected optimization times, we get $O\big(n^{k+1} + \mu n \log n\big)$ as upper bound for both algorithms. □

After seeing that f_k is of reasonable difficulty we continue with our original goal. We demonstrate that A_{spa} needs to have a sufficiently large maximal lifespan in order to be successful on f_k given that the size of the collection of search points μ is (as in the previous section) reasonably bounded.

Theorem 8.12 (Horoba et al. (2009)). *Let $\mu = O\big(n^k/\log n\big)$, $\tau_{\max} = o\big(n^k \log n\big)$, and $k = O(1)$. The optimization time of A_{spa} on f_k is*

$$T_{A_{spa}, f_k} = 2^{\Omega(n^{\varepsilon})}$$

($0 < \varepsilon < 1$ constant) with probability at least $1 - 2^{-\Omega\left(n^{1-o(1)}\right)}$.

Proof. We follow the lines of thought of the proofs of Theorem 8.3 and Theorem 8.11. In particular we use the notation introduced in Theorem 8.11 and consider $T_{A_{\text{spa}}, f_k} = T_{P_1} + T_{P_2} + T_{P_3}$. Since here we want to prove a lower bound on T_{A_{spa}, f_k}, we can estimate $T_{P_1} \geq 0$ and $T_{P_2} \geq 0$ and concentrate on T_{P_3}, only.

Initially we have $\text{OneMax}(x) \leq (2/3)n$ for all $x \in C_0$ with probability $1 - 2^{-\Omega(n)}$ (applying Chernoff bounds for a single search point and the union bound for the collection of search points). Then only mutations further decreasing the number of 1-bits or mutations leading to P_3 are accepted. The probability for a mutation that flips k bits simultaneously is bounded above by $\binom{n}{k}(1/n)^k < 1/(k!)$. We conclude that with probability $1 - 2^{-\Omega(n)}$ we have $\text{OneMax}(x) < (3/4)n$ for the first t with $C_t \cap P_3 \neq \emptyset$. Thus, roughly speaking at least $c \cdot n$ 'jumps' over 'gaps' of size k are needed, where $c > 0$ is some constant. Strictly speaking this is not true. We could also have 'jumps' over 'gaps' of size $2k$, $3k$, … With increasing size of the 'gap' the number of such 'jumps' decreases. Since the probability for 'jumping over gaps' of size s is bounded above by $1/n^s$ and since we do not make any assumption on the size of the constant $c > 0$, we can as well consider $c \cdot n$ 'jumps' over 'gaps' of size k.

For the rest of the proof, we adapt the argumentation from Theorem 8.3. Here, the probability for a specific k-bit mutation in r independent trials is bounded above by $1 - \left(1 - 1/n^k\right)^r$. Considering $r = \alpha(n)(n^k - 1)\ln n$ for some α with $\lim_{n\to\infty} \alpha(n) = 0$, we get $1 - 1/n^{\alpha(n)}$ for the probability to 'jump over' one of the $\Theta(n)$ 'gaps' before the collection of search points gets instinct. Moreover, we get a probability of at most $2^{-\Theta\left(n^{1-\alpha(n)}\right)}$ that this happens at all $\Theta(n)$ 'gaps'. The application of a union bound to the number of steps yields the theorem. $\square$

8.1.4. Remarks on Evolutionary Aging

In contrast to static pure aging, in evolutionary aging an offspring is assigned age 0 independently of its function value. Thus, a copy of a current best search point ensures that the collection of search points does not get extinct. Therefore, we do not need to consider the time until an improvement of the current best function value but rather the time until a copy is made. This is much faster and allows for a smaller maximal lifespan. We investigate this briefly on LeadingOnes and compare it with the bounds derived for static pure aging (Theorem 8.1, Corollary 8.2, and Theorem 8.3).

Theorem 8.13. *Let $\mu = n^{O(1)}$ and $\tau_{\max} = \omega(\mu \log \mu)$. The expected optimization time of A_{eva} on LeadingOnes is*

$$E(T_{A_{eva},\textsc{LeadingOnes}}) = O\left(n^2 + \mu n \log n\right)$$

Proof. As before, we only need to consider the set of currently best search points and show that they are not all removed due to old age. We revisit the proof of Theorem 8.1 and consider a point in time, when the current best function value is increased. Clearly, the improved search point gets age 0. The expected time until the whole collection of search points consists of copies of this current best is $O(\mu \log \mu)$. Due to the coupon collector theorem (Lemma B.17), we see that the probability that this has not happened after $\omega(\mu \log \mu)$ steps, is at most $n^{-\omega(1)}$. Clearly, all search points at this point of time have different age since each copy starts with age 0. Thus, in each following iteration at most one search point can be removed due to old age. Due to this removal worse search points might again enter the collection of search points, but, however, they will quickly be removed by subsequent copies. It follows, that we have always $\Omega(\mu)$ best search points left. In this situation, we get an improved search point with probability $\Omega(1/n)$.

A simple union bound for the at most n needed improvements yields that with probability $1 - n/n^{\omega(1)} = 1 - n^{-\omega(1)}$, A_{eva} manages to find the global optimum of LeadingOnes in time $O\left(n^2 + \mu n \log n\right)$. Since in expectation, we need less than one repetition, the claimed expected optimization time follows. $\square$

We see that evolutionary aging is efficient for much smaller maximal lifespans than static pure aging is. We have shown that static pure aging is inefficient on LeadingOnes if $\tau_{\max} = o(n \log n)$ while $\tau_{\max} = \omega(\mu \log \mu)$ suffices for evolutionary aging. This implies that for smaller sizes of the collection of search points, evolutionary aging still works for parameter setting where static pure aging fails with high probability. In particular, for $\mu = \Theta(n/\log n)$ for evolutionary aging $\tau_{\max} = \omega(n)$ is sufficient whereas for static pure aging $\tau_{\max} = o(n \log n)$ is insufficient. For $\mu = O(1)$, we only need $\tau_{\max} = \omega(1)$ in the case of evolutionary aging. We investigate this setting in the experimental section at the end of this chapter.

8.2. Where a Large Maximal Lifespan Can Be Harmful

In the preceding section we pointed out situations in which a large maximal lifespan is needed. However, aging comes only into play when the maximal lifespan is reached and thus, the maximal lifespan needs to be reasonable if aging is to be effective. If parametrized appropriately, aging allows for (partial and complete) restarts.

In this section, we consider a situation, where a small maximal lifespan can prevent the algorithm under consideration from getting trapped in parts of the search space that keep it away from the global optimum. To this end, we construct an example function for which the maximal lifespan must not be too large in order to achieve efficient optimization.

Our fitness function primarily contains a path similar to Ridge (Definition 8.9) but leading to a local optimum. The global optimum is situated in a certain distance to the path. Thus, it is more likely to stay on the path than 'jumping' to the global optimum. The path itself consists of two sub-paths that are separated by two 'gaps'. The idea is that reaching the second sub-path traps a search point on a path with exponential length that leads away from the global optimum.

To reach this goal, we use a so-called long k-path (Horn et al. 1994; Rudolph 1996), i. e., a very long path of Hamming neighbors with increasing fitness folded into the Boolean hypercube. Note, that a similar approach was followed when defining a difficult to optimize monotone function in Chapter 6. We adopt the notion of long k-paths as presented by Sudholt (2008) and use $\circ$ to denote the concatenation of two bit vectors.

Definition 8.14 (Sudholt (2008))**.** *Let $k, n \in \mathbb{N} \setminus \{1\}$ with $n/k \in \mathbb{N}$. The* long k-path *P_k^n of dimension n is a sequence of points from $\{0,1\}^n$ defined recursively. Let $P_k^0 := ()$, the empty bit sequence, and $P_k^{n-k} = (p_1, \dots, p_\ell)$. Then,*

- $S_0 = (0^k \circ p_1, 0^k \circ p_2, \dots, 0^k \circ p_\ell)$
- $S_1 = (1^k \circ p_\ell, 1^k \circ p_{\ell-1}, \dots, 1^k \circ p_1)$
- $B = (0^{k-1}1 \circ p_\ell, 0^{k-2}1^2 \circ p_\ell, \dots, 01^{k-1} \circ p_\ell)$

and P_k^n is the concatenation of S_0, B, and S_1, i. e., $P_k^n = (S_0, B, S_1)$.

Note, that the search points in S_0 and S_1 differ in the k leading bits and the search points in B represent a bridge between them. Thus, all points are pairwise different. An important property of these paths is that for all $i < k$ and each point p_j on the path with at least i successors on the path, the Hamming distance $H(p_j, p_{j+i})$ equals i. Moreover, for $i \geq k$, $H(p_j, p_{j+i}) \geq k$ holds. Thus, in order to take a large shortcut, a k-bit mutation is necessary. For $k = \Omega(\sqrt{n})$ the probability for such an event is exponentially small. The length of a long k-path of dimension n is $|P_k^n| = k \cdot 2^{n/k} - k + 1$ (Sudholt 2008), which is still exponential for $k = O(\sqrt{n})$. Recall, that we already discussed results on long k-paths in Section 6.1.

As the long k-path has increasing fitness, a search point of A_{spa} that reaches the path will not be eliminated before the end of the path is found provided that it achieves sufficient progress. Thus, if all search points of the collection of search points reach the path, A_{spa} will with overwhelming probability have an exponential optimization time.

The fitness function g_{k_1,k_2,k_3} defined below is essentially a combination of RIDGE with a long k-path for $k = \sqrt{n/4}$. Strictly speaking we need $k = \sqrt{n/4} \in \mathbb{N}$ and $n/k \in \mathbb{N}$. We spare ourselves the additional hassle of using $\left\lfloor \sqrt{n/4} \right\rfloor$ or restricting n to $n = 4i^2$ for $i \in \mathbb{N}$ as this does not lead to significant changes. A visualization of g_{k_1,k_2,k_3} can be found in Figure 8.3.

Definition 8.15 (Horoba et al. (2009))**.** *Define $x^{(\ell)} = (x[0], \dots, x[3n/4-1])$ and $x^{(r)} = (x[3n/4], \dots, x[n-1])$. For n, k_1, k_2, $k_3 \in \mathbb{N} \setminus \{1\}$ with k_1, k_2, $k_3 = O(1)$ and $1 < k_1 < k_2$ we define $g_{k_1,k_2,k_3} \colon \{0,1\}^n \to \mathbb{R}$ by*

$$g_{k_1,k_2,k_3}(x) := \begin{cases} 2^n & \text{if } x = 1^{2n/3}0^{n/3-k_3}1^{k_3} \\ n^3 + i & \text{if } x = p'_i \\ n^2 + n & \text{if } x = 1^{3n/4-k_2}0^{n/4+k_2} \\ n^2 + i & \text{if } x = r_i \\ n - \text{ONEMAX}(x) & \text{otherwise} \end{cases}$$

with

$$p'_i = x^{(\ell)} \circ x^{(r)}, x^{(\ell)} = 1^{3n/4}, x^{(r)} = p_i \in P^{n/4}_{\sqrt{n/4}}$$
$$r_i = 1^i 0^{n-i}, 0 \le i \le 3n/4 - k_1 - k_2.$$

Note, that the p'_i represent the points on the long k-path for $k = \sqrt{n/4}$, whereas the r_i are points on the ridge. There are two 'gaps' of size k_1 and k_2, respectively, in the path leading to the long k-path. The first search point 'jumping over' the first 'gap' is likely to take over the whole collection of search points (provided that μ is not too large) with copies of itself. Then the whole collection of search points has identical age and gets extinct completely due to its age in a single iteration given that the maximal lifespan $\tau_{\max}$ is sufficiently small. This is equivalent to a restart. We do not claim that such a 'double gap' is necessary for aging to be effective. It does, however, simplify the proof considerably. We remark that the combination of restarts with two paths that are entered with different probabilities has already been presented by Jansen (2002).

Since we use g_{k_1,k_2,k_3} always with the same parameters k_1, k_2, and k_3, we write g instead of g_{k_1,k_2,k_3} in the following. Analogous to Section 8.1.3 we partition the search

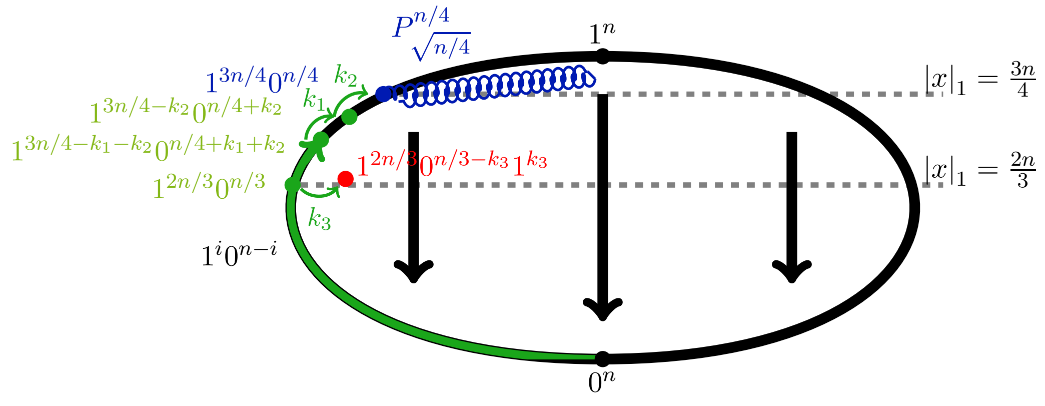


Figure 8.3.: The example function g.

space $\{0,1\}^n$ into the following five disjoint sets.

$$\begin{aligned}
P_2 &:= \{r_i \mid 0 \leq i < 3n/4 - k_1 - k_2\} \\
P_3 &:= \{r_{3n/4-k_1-k_2}, 1^{3n/4-k_2}0^{n/4+k_2}\} \\
P_4 &:= \{p_i' \mid p_i \in P^{n/4}_{\sqrt{n/4}}\} \\
P_5 &:= \{1^{2n/3}0^{n/3-k_3}1^{k_3}\} \\
P_1 &:= \{0,1\}^n \setminus (P_2 \cup P_3 \cup P_3 \cup P_4)
\end{aligned}$$

The sets P_1, P_2, P_3, P_4, P_5 again form a g-based partition $P_1 <_g P_2 <_g P_3 <_g P_4 <_g P_5$.

A typical run of A_{spa} starts in P_1 and moves towards 0^n until we reach some point in P_2. Then we follow the path defined by the points in P_2 until either the global optimum P_5 is found or the end of the path is reached. In the latter case, the search points may 'jump over' the two 'gaps' (passing P_3) and reach P_4, if the maximal lifespan $\tau_{\max}$ is large enough. Otherwise the collection of search points will become extinct and restart in P_1 until finally the 'jump' to the global optimum P_1 succeeds.

We first show that a standard $(\mu{+}1)$ EA as well as A_{spa} where the maximal lifespan is too large have difficulties in optimizing g as they can easily get trapped on the long k-path.

Theorem 8.16 (Horoba et al. (2009))**.** *Let $k_1, k_2, k_3 \in \mathbb{N}\setminus\{1\}$ with $k_1 < k_2$, $k_1 < k_3$ and $k_1, k_2, k_3 = O(1)$, $\mu = O\big(n^{k_1-1}\big)$, and $\tau_{\max} = \Omega\big(\log n\,(n^{k_2} + \mu \log n)\big) = \Omega\big(n^{k_2}\log n\big)$. The optimization time of A_{spa} on g is*

$$T_{(\mu+1)\text{ EA},\ g} = 2^{\Omega(\sqrt{n})}$$
$$T_{A_{spa},\ g} = 2^{\Omega(\sqrt{n})}$$

with probability $1 - o(1)$ each.

Proof. We use the partition of the search space $\{0,1\}^n$ defined above. Let again T denote the optimization time of the algorithm under consideration. Moreover, we use the notion of T_{P_1}, T_{P_2}, and T_{P_3} introduced in the proof of Theorem 8.11. Let additionally T_{P_4} denote the number of steps where we have $C_t \subseteq P_1 \cup P_2 \cup P_3 \cup P_4$, $C_t \cap P_4 \neq \emptyset$ and $\max\{g(x) \mid x \in C_t\} < \max\{g(x) \mid x \in P_4\}$.

It is easy to see that as soon as the first search point has reached the path, the whole collection of search points will follow. Once all search points are on the long k-path the optimization time is exponential for both algorithms with high probability.

On the long k-path a specific 1-bit mutation suffices to create an offspring with a larger fitness. As noted above, long k-paths are constructed in such a way that a mutation of at least k bits is necessary in order to take a large shortcut. Populations do not increase the probability of such mutations. Thus, the lower bound on the optimization time of the (1+1) EA (Droste et al. 2002) carries over to the (μ+1) EA. For $k = \sqrt{n/4}$, we therefore get $\mathrm{E}(T_{P_4}) = \Omega\Big((n/4)^{3/2}2^{\sqrt{n/4}}\Big)$. Moreover, the probability to have a polynomial optimization time is exponentially small.

It remains to show that the complete collection of search points reaches the long k-path with probability $1-o(1)$. With probability $1-e^{-\Omega(n)}$ we have $\textsc{OneMax}(x) < (2/3)n-\sqrt{n}$ for all $x \in C_0$ (Chernoff bounds) making a direct mutation into the global optimum exponentially unlikely. After that the number of 1-bits can only decrease until some ridge point is found.

For each point in P_2 there is a successor with Hamming distance 1 and larger function value. The Hamming distance to the global optimum is always bounded below by $k_3 > 1$. It equals k_3 for $1^{2n/3}0^{n/3}$ and is strictly larger than that for all other points on the ridge. The probability to 'jump' to the global optimum is $O\big(1/n^{k_3}\big)$. Thus, for $1^{2n/3}0^{n/3}$ the probability to reach the successor prior to 'jumping' to the global optimum is $1-O(1/n)$ and it is $1-O\big(1/n^2\big)$ for all other points in P_2. Note that there are less than n points in P_2. For the first search point we see that we reach the point with maximal function value in P_2 without finding the global optimum with probability $1-O(1/n)$. The i-th search point gets past the global optimum also if it is replaced due to copying a better search point. Since we assume $\mu = O\big(n^{k_1-1}\big)$, this happens with conditional probability

$$\Omega\left(\frac{\frac{1}{n}+\frac{i}{\mu}}{\frac{1}{n^{k_3}}+\left(\frac{1}{n}+\frac{i}{\mu}\right)}\right) = \Omega\left(\frac{\frac{1}{n}+\frac{i}{n^{k_1-1}}}{\frac{1}{n^{k_3}}+\left(\frac{1}{n}+\frac{i}{n^{k_1-1}}\right)}\right)$$
$$= \Omega\left(\frac{n^{k_3-1}+i\cdot n^{k_3-k_1+1}}{n^{k_3-1}+i\cdot n^{k_3-k_1+1}+1}\right) = 1-O\left(\frac{1}{n^{k_3-1}+i\cdot n^{k_3-k_1+1}+1}\right).$$

Since $k_1 < k_3$, this is $1-O\big(1/(in^2)\big)$ and we obtain $1-O\left(1/n+\sum\limits_{i=1}^{\mu-1}1/(in^2)\right)$ $= 1-O(1/n)$ as bound on the probability to pass by without stumbling over the global optimum by accident.

Once this is the case, the long k-path is entered after one mutation of exactly k_1 specific bits followed by a second mutation of exactly k_2 specific bits. Each such mutation has probability $\Omega\big(1/n^{k_2}\big)$. Since now the unique global optimum has Hamming distance $\Theta(n)$ we see that the long k-path is entered with probability exponentially close to 1 given that no search point is removed due to its age. In order to see this we need to consider the time needed to enter the long k-path. We can concentrate on the two 'gaps' of sizes k_1 and k_2 since before we only need to deal with 1-bit mutations and did this already in the proof of Theorem 8.11.

Remember that it takes in expectation $O(\mu \log n)$ steps until the complete collection of search points consists of copies of the current best, provided that the current best does not improve in this time. We have $\mu = O\big(n^{k_1-1}\big)$ and see that the expected number of steps until the complete collection of search points performs one of the two large 'jumps' is bounded by $O\big(n^{k_1-1} \log n\big) + O\big(n^{k_2}\big) = O\big(n^{k_2}\big)$. Since we have $\tau_{\max} = \Omega\big(n^{k_2} \log n\big)$ we see that the complete collection of search points reaches the long k-path with probability $1 - o(1)$. □

We have seen that A_{spa} has difficulties in optimizing g if the maximal lifespan $\tau_{\max}$ is too large. We see that in this situation aging is not effective. In principle it does not come into play at all since no search point reaches the maximal lifespan with high probability. In particular it shows the same behavior as a standard $(\mu{+}1)$ EA for a very long time. This is due to the long k-path where it can be trapped while making little progress. In the following we therefore point out the advantages of A_{spa} when $\tau_{\max}$ is sufficiently small and show that this can prevent the algorithm from being trapped on the long k-path. Note, that the maximal lifespan $\tau_{\max}$ must not be too small, either (see Section 8.1.1). This is an example where aging is beneficial and performs a kind of restart.

Theorem 8.17 (Horoba et al. (2009)). *Let $k_1, k_2, k_3 \in \mathbb{N} \setminus \{1\}$ with $2 < k_1 < k_2$ and $k_1, k_2, k_3 = O(1)$, $\mu = O\big(n^{k_1-1}\big)$, and $\tau_{\max} = \omega\big(\log n \left(n^2 + \mu n \log n\right)\big)$ and $\tau_{\max} = O\big(n^{k_2-k_3}\big)$.*

The optimization time of A_{spa} on g is

$$T_{A_{spa},g} = O\Big(n^{k_1+k_3+1} + \mu n^{k_1+k_3} \log n + n^{k_2-1}\Big)$$

with probability $1 - o(1)$.

Proof. We follow the line of thought from the proof of Theorem 8.16 and use the notation defined there. Note, that the only fundamental difference is the behavior of the algorithm in P_3. Therefore, we concentrate on this and the derivation of T_{P_3} in the following. From the proof of Theorem 8.11 we get $\mathrm{E}\left(T_{P_1}\right) = O(\mu n + n \log n)$.

Remember that the expected time to make μ copies of the current best is $O(\mu \log n)$ and that the expected time needed to reach the best point in P_2 is $O\big(n^2 + \mu n \log n\big)$ (Witt 2006). Since $\tau_{\max} = \omega\big(\log n \left(n^2 + \mu n \log n\right)\big)$ is sufficiently large we conclude that after $\mathrm{E}\left(T_{P_2}\right) = O\big(n^2 + \mu n \log n\big)$ we have a collection of search points that contains only search points with function values that are at least as large as that of the best point in P_2.

Pessimistically we assume that all members of the collection of search points are equal to the best point in P_2. In this situation the first 'gap' of size k_1 is 'jumped over' with probability $\Theta\big(1/n^{k_1}\big)$ in the next iteration. If this does not happen (since, for example, $\tau_{\max}$ is too small) the search points in the collection of search points may eventually be removed due to their age. In any case, we are not worse than if the collection of search points was started again. Note that we do not assume to have a complete restart in a single iteration.

Recall that $O\big(n^2 + \mu n \log n\big)$ is an upper bound on the number of iterations needed to get into this situation. Then we have a probability of $\Theta\big(1/n^{k_1}\big)$ to increase the function value. We conclude that on average this happens after $O\big(n^{k_1+2} + \mu n^{k_1+1} \log n\big)$ iterations. Note that this search point has age 0 in this situation since it improved over its parent. Thus, after another $O(\mu \log n) = O\big(n^{k_1-1} \log n\big)$ iterations on average the collection of search points consists of copies of this one search point all with the very same age. We conclude that we get in this situation with probability $1 - O(\log(n)/n)$ since the only reason to have different ages is to have another search point make this 'jump' which only occurs with probability $O\big(1/n^{k_1}\big)$.

The probability to make the second 'jump' is bounded above by $O\big(1/n^{k_2}\big)$. Remember that we have $k_1 < k_2$ and $\tau_{\max} = O\big(n^{k_2-k_3}\big)$. Thus, with probability $1-O\big(1/n^{k_3}\big)$ before this happens the complete collection of search points is removed in a single iteration due to its age. This, clearly, is completely equivalent to a restart. We see that with probability $1 - O(\log n/n) - O\big(1/n^{k_3}\big)$ we have something that equals a restart every $O\big(n^{k_1+2} + \mu n^{k_1+1} \log n + \tau_{\max}\big) = O\big(n^{k_1+2} + \mu n^{k_1+1} \log n + n^{k_2-k_3}\big)$ iterations since we need to wait for the search points to reach age $\tau_{\max}$.

While A_{spa} moves through P_2 for each point in P_2 we have that it becomes current best point in the collection of search points at some point with probability $1 - O(1/n)$. This holds since mutations 'jumping over' any such point have probability $O\big(1/n^2\big)$ while they reach it with probability $\omega(1/n)$. Since the expected time to move the complete collection of search points beyond a point that once was current best is trivially bounded below by $\Omega(\mu)$, we have that the probability that a current best member of a collection of search points is selected as parent at least once is bounded below by $\Omega(1)$. We consider $1^{2n/3}0^{n/3} \in P_2$ and see that it has Hamming distance k_3 to the global optimum. Clearly, each point in P_2 becomes member of the current collection of search points while moving through P_2 with probability $\Omega(1)$. Thus, while moving through P_2 the probability to find the global optimum is bounded below by $\Omega\big(1/n^{k_3-1}\big)$. The -1 in the exponent is due to the fact that any mutation increasing the function value has probability $O(1/n)$. Thus, on average after $O\big(n^{k_3-1}\big)$ times A_{spa} moves through P_2 the optimum is found.

We combine what we have and see that with probability $1-o(1)$ A_{spa} find the optimum of g within

$$O\Big(n^{k_3-1} \cdot \Big(n^{k_1+2} + \mu n^{k_1+1} \log n + n^{k_2-k_3}\Big)\Big) = O\Big(n^{k_1+k_3+1} + \mu n^{k_1+k_3} \log n + n^{k_2-1}\Big)$$

iterations. This holds with probability $1 - o(1)$ since the probability to have a successful 'jump' over the second 'gap' in n^{k_3-1} rounds is $o(1)$ due to our assumptions: Having $\tau_{\max} = \omega\big(\log n\, \big(n^2 + \mu n \log n\big)\big)$ and $\tau_{\max} = O\big(n^{k_2-k_3}\big)$ implies that $k_3 < k_2 - 2$. □

8.3. Combining Example Functions

Within this section, we develop a general technique that can be used to strengthen the properties of an example function or to combine the properties of different example functions. We apply this technique to derive an example function that combines the properties of the two functions discussed in the previous sections. This results in a function where the maximal lifespan has to be chosen within an arbitrarily small interval.

Consider k pseudo-Boolean functions $f_i\colon \{0,1\}^{n_i} \to \mathbb{R}^+$ with $i \in \{1,\dots,k\}$. Choose f_i^{ub} with $f_i^{\mathrm{ub}} \geq \max\{f_i(x^{(i)}) \mid x^{(i)} \in \{0,1\}^{n_i}\}$. Finding such an upper bound is unproblematic if we consider example functions. We intend to combine all f_i into a single pseudo-Boolean function. The basic idea is to create a function that drives an algorithm into optimizing all f_i sequentially.

We consider two alternatives to *finish* the optimization of a f_i: Finding a search point $x^{(i)}$ with $x^{(i)} \in \mathrm{OPT}_i$ or $x^{(i)} \in \mathrm{TRAP}_i$ where OPT_i and TRAP_i are two disjoint subsets of the search space $\{0,1\}^{n_i}$. In the first case we call the optimization of f_i a *success* and in the second case we call the optimization of f_i a *failure*.

Consider $h\colon \{0,1\}^n \to \mathbb{R}^+$ where $n = \sum_{i=1}^{k} n_i$. We partition a bit vector $x = (x[0],\dots,x[n-1])$ of length n into k bit vectors

$$x^{(i)} = \left(x\left[\sum_{j=1}^{i-1} n_j\right],\dots,x\left[\sum_{j=1}^{i} n_j - 1\right]\right)$$

of length n_i to ease the understanding of the function definition

$$h(x) = \begin{cases} \sum\limits_{i=1}^{a} f_i^{\mathrm{ub}} + f_{a+1}(x^{(a+1)}) & \text{if } a < k \\ \sum\limits_{i=1}^{a} f_i^{\mathrm{ub}} + 1 & \text{if } a = k \text{ and } b \leq \lfloor k/2 \rfloor \\ \sum\limits_{i=1}^{a} f_i^{\mathrm{ub}} + 2 & \text{if } a = k \text{ and } b > \lfloor k/2 \rfloor \end{cases}$$

where

$$a = \max\Big\{1 \leq i \leq k \mid \forall 1 \leq j \leq i\colon x^{(j)} \in \mathrm{OPT}_j \cup \mathrm{TRAP}_j\Big\}$$

is the maximal length of a sequence $f_1,\dots,f_a$ of finished f_i and

$$b = \left|\Big\{1 \leq i \leq k \mid x^{(i)} \in \mathrm{OPT}_i\Big\}\right|$$

is the number of successfully finished f_i. A search point $x = (x^{(1)},\dots,x^{(k)})$ constitutes a global optimum of h iff $f_i(x^{(i)}) \in \mathrm{OPT}_i \cup \mathrm{TRAP}_i$ for all i and $f_i(x^{(i)}) \in \mathrm{OPT}_i$ for more than $\lfloor k/2 \rfloor$ different i.

An application of this technique leads to the creation of an example function that combines the properties of the example function f_k from Definition 8.10 and the example function g from Definition 8.15. Consider $f_k\colon \{0,1\}^{n/2} \to \mathbb{R}^+$ as f_1 and $g\colon \{0,1\}^{n/2} \to \mathbb{R}^+$ as f_2. Choose $f_1^{\mathrm{ub}} = n/2 + \lfloor n/(2k) \rfloor$ and $f_2^{\mathrm{ub}} = 2^{n/2}$. Let $\mathrm{OPT}_1 = \{1^{\lfloor n/(2k) \rfloor k} 0^{n/2 - \lfloor n/(2k) \rfloor k}\}$

and $\mathrm{TRAP}_1 = \emptyset$ as well as $\mathrm{OPT}_2 = \{1^{2n/6}0^{n/6-k_3}1^{k_3}\}$ and $\mathrm{TRAP}_2 = \emptyset$. We combine f_1 and f_2 using the approach explained above. We refer to the resulting function as h_{k,k_1,k_2,k_3} and again simply write h.

The following theorem shows that A_{spa} does not optimize h efficiently if $\tau_{\max}$ is chosen too small or too big. If $\tau_{\max}$ is chosen within a certain interval, A_{spa} efficiently optimizes$h$.

Theorem 8.18 (Horoba et al. (2009)). *Let $k, k_1, k_2, k_3 \in \mathbb{N} \setminus \{1\}$ with $k, k_1, k_2, k_3 = O(1)$, $k < k_1 < k_2$, $k_1 < k_3$, and $k_3 < k_2 - 2$ as well as $\mu = O\big(n^k/\log n\big)$. Then, the following holds.*

1. *If $\tau_{\max} = o\big(n^k \log n\big)$:*
$$T_{A_{spa},\,h} = 2^{\Omega(n^{\varepsilon})}$$
with probability $1 - 2^{-\Omega\left(n^{1-o(1)}\right)}$ for all constants $0 < \varepsilon < 1$.

2. *If $\tau_{\max} = \omega\big(n^k \log n + \mu n \log^2 n\big)$ and $\tau_{\max} = O\big(n^{k_2-k_3}\big)$:*
$$T_{A_{spa},\,h} = O\Big(n^{k_1+k_3}\,(n + \mu \log n) + n^{k_2-1}\Big)$$
with probability $1 - o(1)$.

3. *If $\tau_{\max} = \Omega\big(n^{k_2} \log n\big)$:*
$$T_{A_{spa},\,h} = 2^{\Omega(\sqrt{n})}$$
with probability $1 - o(1)$.

Proof. We proof the three statements from the theorem separately.

1. Due to Theorem 8.12, A_{spa} does not finish f_k within $2^{\Omega(n^{\varepsilon})}$ iterations with probability $1 - 2^{-\Omega\left(n^{1-o(1)}\right)}$ for all constants $0 < \varepsilon < 1$. Hence, the optimization time of A_{spa} on h is $2^{\Omega(n^{\varepsilon})}$ with probability $1 - 2^{-\Omega\left(n^{1-o(1)}\right)}$.

2. Due to Theorem 8.11, A_{spa} finishes f_k within an expected number of $O\big(n^{k+1}\big)$ iterations. Hence, the optimization of f_k is finished after $O\big(n^{k+2}\big)$ iterations with probability $1 - 2^{-\Omega(n)}$. After the first search point x has finished f_k, the second half of x is uniformly distributed. From now on x optimizes g. Due to the proof of Theorem 8.17, with probability $1 - o(1)$ A_{spa} successfully finishes g after $O\big(n^{k_3-1}\big)$ attempts, which result in restarting the algorithm or finding the optimum. Note, that all attempts require $O\big(n^{k_1+2} + \mu n^{k_1+1} \log n + n^{k_2-k_3}\big)$ iterations with probability $1 - o(1)$. Hence, the optimization time of A_{spa} on h is
$$O\Big(n^{k_3-1} \cdot \Big(n^{k+2} + n^{k_1+2} + \mu n^{k_1+1} \log n + n^{k_2-k_3}\Big)\Big) = O\Big(n^{k_1+k_3+1} + \mu n^{k_1+k_3} \log n + n^{k_2-1}\Big)$$
with probability $1 - o(1)$.

3. After the first search point x has finished f_k, the second half of x is uniformly distributed. From now on x optimizes g. Due to Theorem 8.16, A_{spa} does not finish g within $2^{\Omega(\sqrt{n})}$ iterations with probability $1-o(1)$. Hence, the optimization time of A_{spa} on h is $2^{\Omega(\sqrt{n})}$ with probability $1-o(1)$. □

Consider $k = 2$, $k_1 = 3$, $k_2 = 7$, and $k_3 = 4$. Due to Theorem 8.18 we have to choose a $\tau_{\max}$ with $\tau_{\max} = \omega\big(n^k \log n + \mu n \log^2 n\big)$ and $\tau_{\max} = O\big(n^{k_2-k_3}\big)$ to guarantee polynomial optimization time with probability $1-o(1)$. Plugging in the above values yields $\tau_{\max} = \omega\big(n^2 \log n + \mu n \log^2 n\big)$ and $\tau_{\max} = O\big(n^3\big)$. Let $\mu = \Theta\big(n^{2-\varepsilon}/\log^2 n\big)$. Then, the requirements reduce to $\omega\big(n^{3-\varepsilon}\big)$ and $O\big(n^3\big)$ for an arbitrarily small constant $0 < \varepsilon < 1$. We conclude that an appropriate choice of $\tau_{\max}$ can be very difficult. Note, that for any constant $k > 1$ we can obtain the same result by setting $k_1 = k+1$, $k_2 = k_1 + 4$, $k_3 = k_1 + 1$, and having $\mu = \Theta\big(n^{k-\varepsilon}/\log^2 n\big)$. Thus, the small band of appropriate values for $\tau_{\max}$ can be shifted.

8.4. Experimental Supplements

In the preceding sections we have analyzed the influence of the parameter $\tau_{\max}$ for the static pure aging operator and briefly pointed out differences to evolutionary aging. We have shown for which setting of the parameter A_{spa} yields expected polynomial optimization time and for which it yields exponential optimization time with high probability. However, our bounds on $\tau_{\max}$ are not tight. Therefore, we supplement our theoretical results by experiments shedding light on the robustness of the considered algorithm with respect to the maximal lifespan $\tau_{\max}$.

We perform experiments for A_{spa} on LeadingOnes and OneMax as well as A_{eva} on LeadingOnes. Note, that we refrain from doing experiments for f_k and g. This has several reasons. An experimental analysis for f_k would be very similar to the one we execute for LeadingOnes and OneMax. We prefer these two easy functions since they were analyzed in previous chapters. Moreover, the optimization time of f_k increases exponentially with k such that only small values for k are reasonable for experiments. However, for these values the function is still quite similar to LeadingOnes. Thus, we believe that an experimental analysis of f_k would not add additionally insights.

Things are even worse for g since in order to meet the requirements of the theorems we need large values for the parameters of g. Theorem 8.16 requires at least $k_1 = 2$, $k_2 = 3$, and $k_3 = 3$, Theorem 8.17 at least $k_1 = 3$, $k_2 = 6$, and $k_3 = 3$, and Theorem 8.18 at least $k = 2$, $k_1 = 3$, $k_2 = 7$, and $k_3 = 4$. This yields an upper bound of at least $O\big(n^6\big)$.

Moreover, in order to execute sensible experiments, we need very large values of n. Recall that the length of a long k-path of dimension n is $|P_k^n| = k \cdot 2^{n/k} - k + 1$ (Sudholt 2008) and that we consider $P^{n/4}_{\sqrt{n/4}}$. Clearly, we need the length of the path to be larger than the expected optimization time of $O\big(n^6\big)$ as seen above. This results in $n > 16{,}000$, making a sensible experimental analysis very hard. We remark that these two functions were carefully constructed in order to prove certain properties of the operators and thus, are in particular of theoretical interest.

The experiments done in the following can be separated into two sets. First, we consider parameter settings which yield polynomial upper bounds on the expected optimization time (Corollary 8.2 and 8.5). In order to analyze the effects of the size of the collection of search points all sets of experiments are performed for $\mu \in \{1, \lfloor\sqrt{n}\rfloor, n\}$ where $\mu = 1$ leads to the special case of the (1+1) EA. The choice $\mu \approx \sqrt{n}$ has often turned out to be a good choice (Harik et al. 1999). Moreover we are interested in the effects of bigger collections of search points sizes, i. e., sizes that are not sublinear. Since the size of the collection of search points has an enormous effect on the optimization time, we chose $\mu = n$ for this purpose. As maximal lifespan $\tau_{\max}$ we use the values which are bounded below by the our results. For a bound $\omega(b(n))$ we set $\tau_{\max} = \lfloor 6\log(n) \cdot b(n)\rfloor$, where the 6 helps for small values of n. For each of these experiments we perform 100 independent runs of the considered algorithm and plot the results using box-and-whisker plots (Definition B.1) for $n \in \{10, 20, \ldots, 200\}$. For the sake of comparison we present results for the (μ+1) EA, i. e., the corresponding algorithm without aging.

Second, we analyze the influence of the maximal lifespan $\tau_{\max}$ for $n \in \{10, 20, \ldots, 200\}$ and the same settings of μ as in the first set of experiments. Since in the first set of experiments $\tau_{\max}$ was chosen according to the theorems, we now perform experiments with stepwise decreasing values for $\tau_{\max}$. For each value of n and μ we perform experiments with 20 equidistantly chosen values for $\tau_{\max}$, i. e., $\tau_{\max} = (i/20) \cdot \tau_{\max}$ ($i \in \{1, \ldots, 20\}$) where $\tau_{\max}$ is the corresponding $\tau_{\max}$ value from the first set of experiments. We fix the maximal number of iterations executed to the corresponding upper quartile from the (μ+1) EA. The results are given as a 3D plot for each μ showing the number of successful runs within 100 independent runs for all pairs of n and $\tau_{\max}$. In these plots the number of successful runs is mapped to different colors that are projected onto the horizontal plane to improve visibility. Note that the results of these experiments need to be analyzed very carefully as with the setting used it is not possible to make conclusions about the 'true' order of magnitude needed for $\tau_{\max}$. From a formal point of view $(i/20) \cdot \tau_{\max}$ can be seen as $\Theta(\tau_{\max})$ for each value of $i \in \{1, \ldots, 20\}$.

8.4.1. Results for Static Pure Aging

The results for LeadingOnes and OneMax for the first set of experiments can be found in Figure 8.4 and Figure 8.5, respectively. In both figures, results for A_{spa} and the (μ+1) EA are displayed next to each other for comparison. Obviously, the use of aging does not effect the optimization time of the algorithm since both algorithms perform equally for the considered input sizes, also with respect to the leading constants. Note, that for A_{spa}, we used $\tau_{\max} = \left\lfloor 6\log^2 n(n + \mu\log n)\right\rfloor$ for LeadingOnes (Corollary 8.2) and $\tau_{\max} = \lfloor 6\log n(\mu\log\mu + n)\rfloor$ for OneMax (Corollary 8.5).

The results for the second set of experiments are depicted in one joint diagram for LeadingOnes and OneMax (Figure 8.6). Note that we expect 75 successful runs within 100 runs as we fix the maximal number of iterations executed to the corresponding upper quartile from the first set of experiments. We see that for both functions the algorithm is quite robust with respect to the maximal lifespan. In particular on LeadingOnes it performs quite well for a large range of values for $\tau_{\max}$ (independently of the size of the collection of search points). Only for very small values of $\tau_{\max}$, we have a small success probability. For OneMax, the robustness increases with increasing size of the collection of search points.

For the sake of visibility we plot the success rate for the smallest $\tau_{\max}$ tried, namely $\tau_{\max} = n$ for LeadingOnes (Theorem 8.3) and $\tau_{\max} = n^{0.9}$ for OneMax (Theorem 8.6, setting δ somewhat arbitrarly). For these values of $\tau_{\max}$, we proved an exponential lower bound on the optimization time with high probability and we see that indeed the success rate drops drastically already for small values of n.

8.4.2. Results for Evolutionary Aging

The results for evolutionary aging can be found in Figure 8.8 and Figure 8.9, respectively. We first do experiments for A_{eva} on LeadingOnes with $\tau_{\max} = \lfloor 6 \log(n) \mu \log \mu \rfloor$. Since in the case $\mu = 1$, this becomes zero, we choose $\tau_{\max} = \lfloor 6 \log n \rfloor$ instead. We see again, that the optimization times observed during the experiments corresponds to the results of the (μ+1) EA. Thus, in fact, for the (1+1) EA a maximal lifespan in order $O(\log n)$ suffices.

For the second set of experiments we perform experiments for different maximal lifespans where the smallest $\tau_{\max}$ considered is $\tau_{\max} = 3$. We see that evolutionary aging is, at least for small collections of search points, not as robust with respect to the maximal lifespan as static pure aging. With increasing n, the range in which we observe good success rates, becomes quite small. We further investigate this aspect within the next chapter, where we compare the benefits and drawbacks of both aging mechanisms.

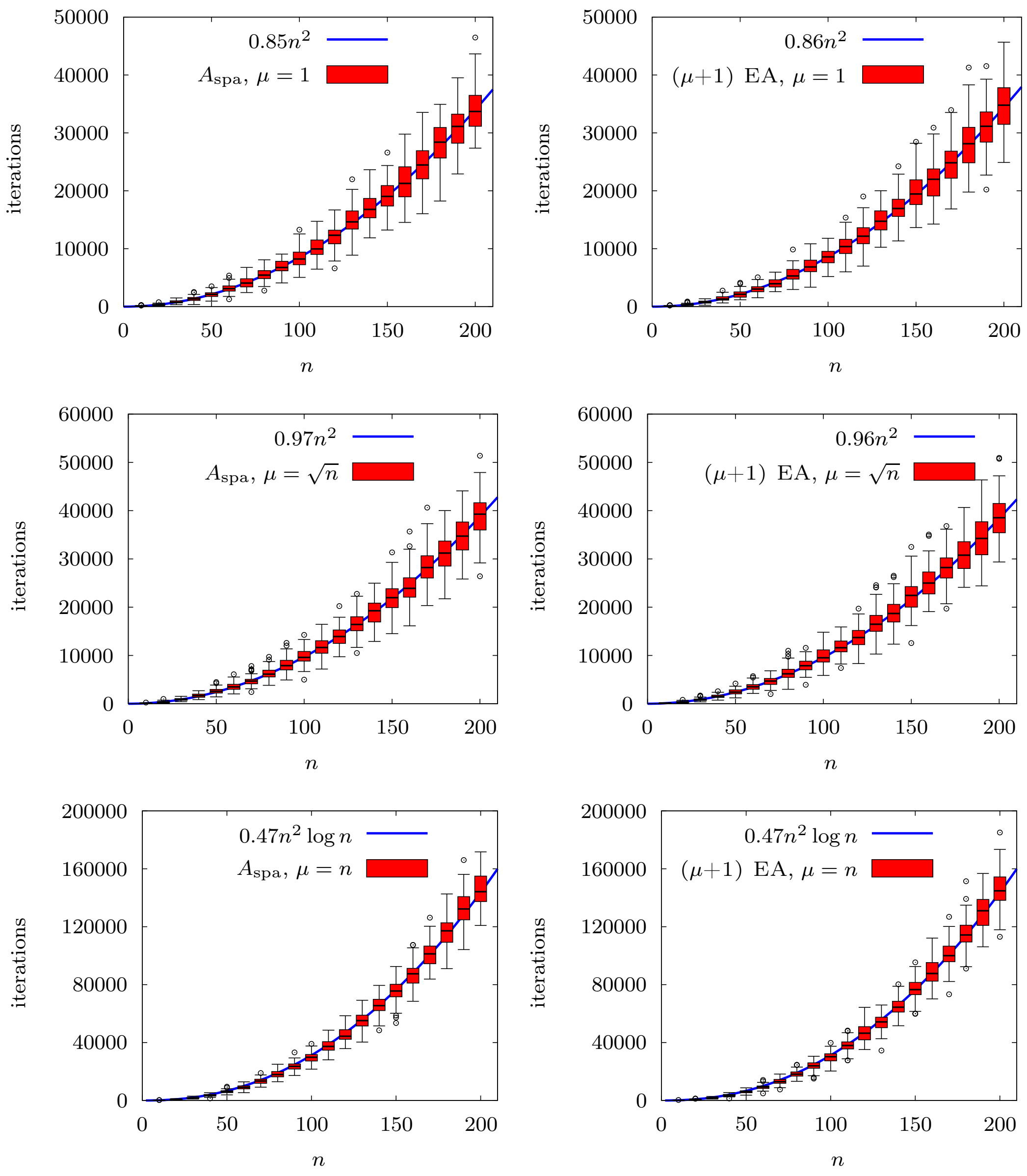


Figure 8.4.: Optimization times of A_{spa} and the $(\mu+1)$ EA on LeadingOnes with different sizes of the collection of search points, data from 100 runs.

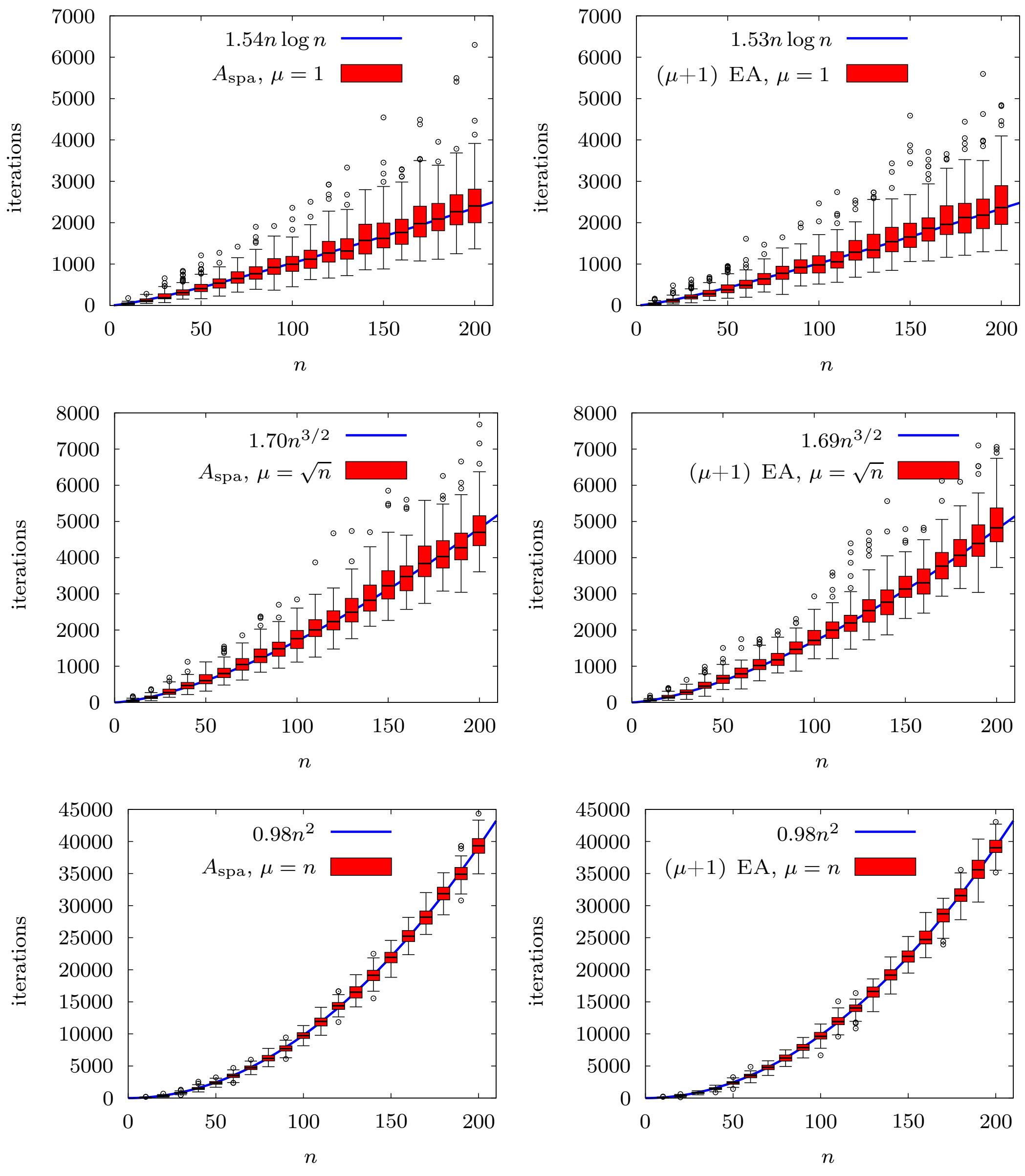


Figure 8.5.: Optimization times of A_{spa} and the (μ+1) EA on OneMax with different sizes of the collection of search points, data from 100 runs.

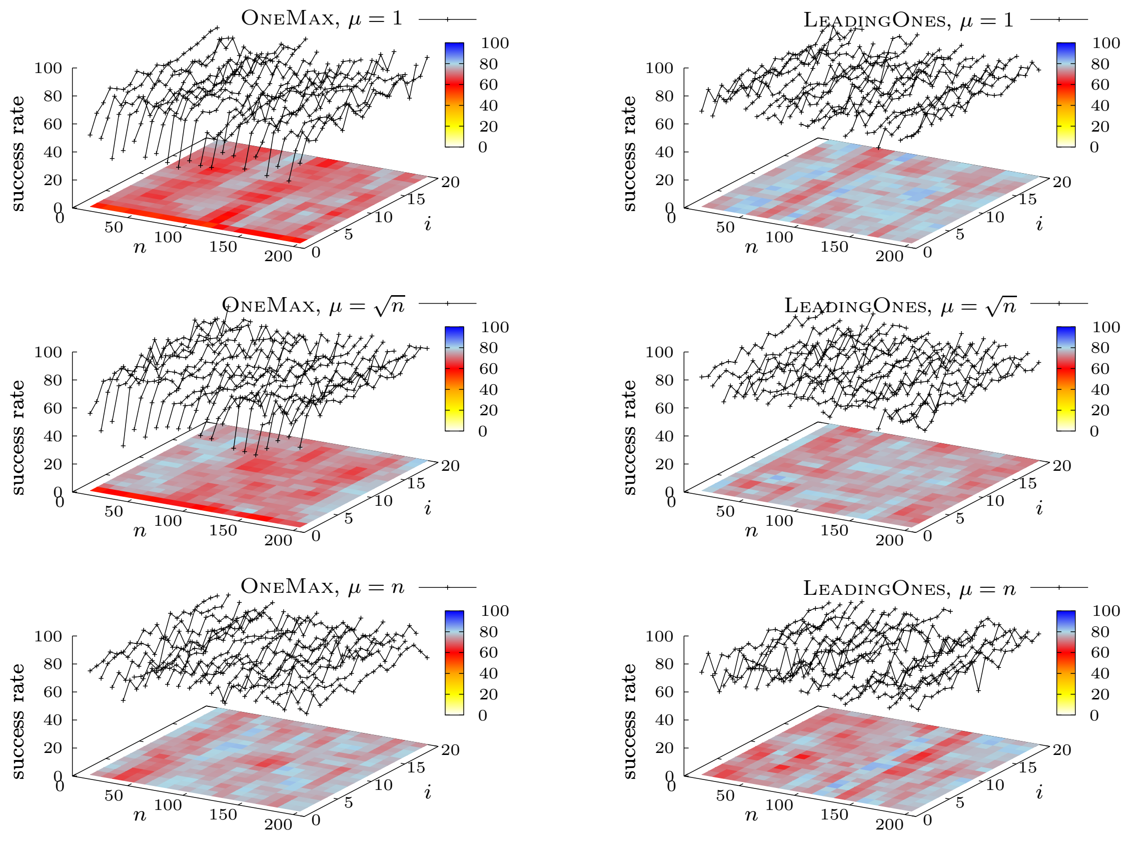


Figure 8.6.: Success rates in 100 independent runs of A_{spa} on OneMax and LeadingOnes with different sizes μ of the collection of search points and decreasing values of $\tau_{\max}$.

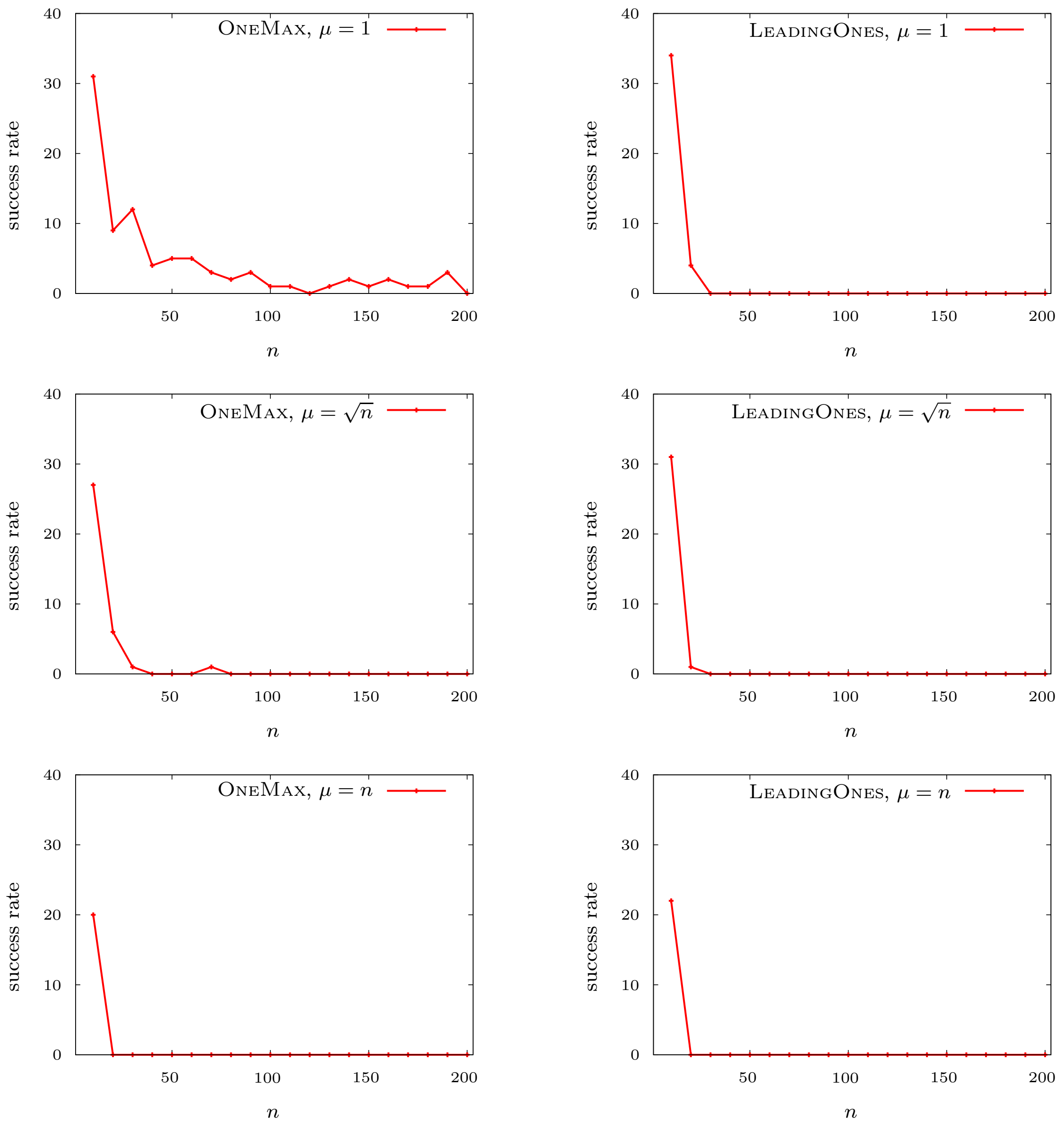


Figure 8.7.: Success rates in 100 independent runs of A_{spa} on ONEMAX and LEADINGONES with different sizes μ of the collection of search points and $\tau_{\max} = n^{0.9}$ and $\tau_{\max} = n$, respectively.

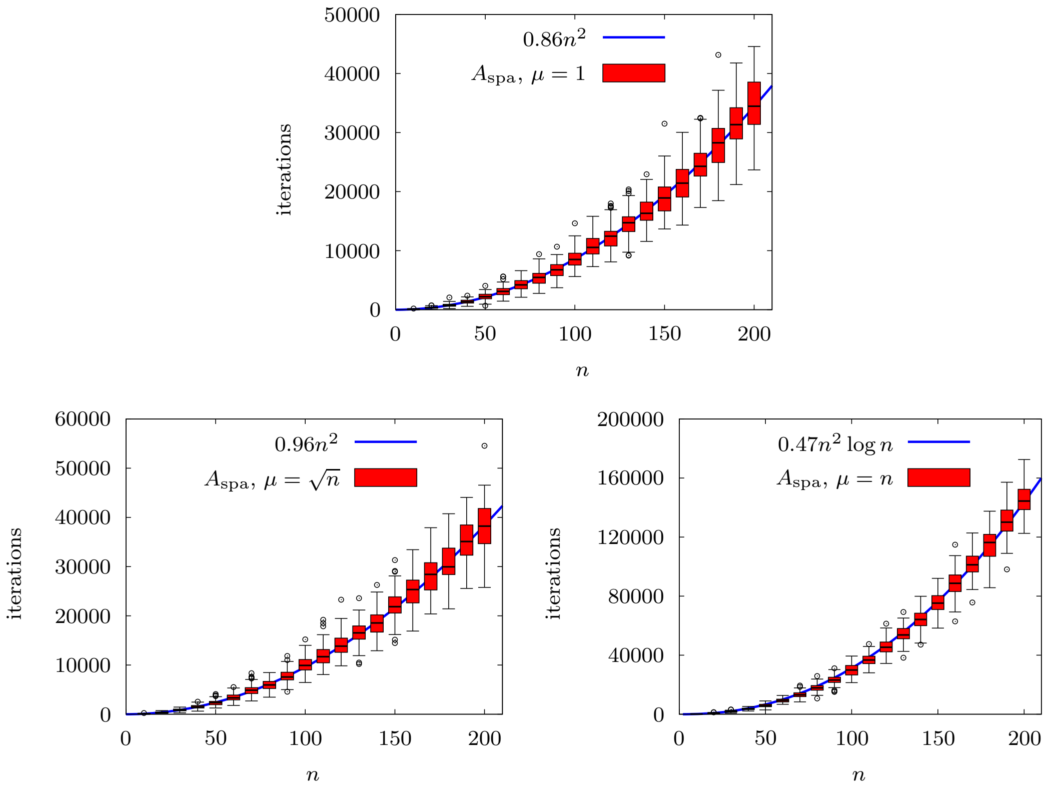


Figure 8.8.: Optimization times of A_{eva} on LEADINGONES with different sizes of the collection of search points, data from 100 runs.

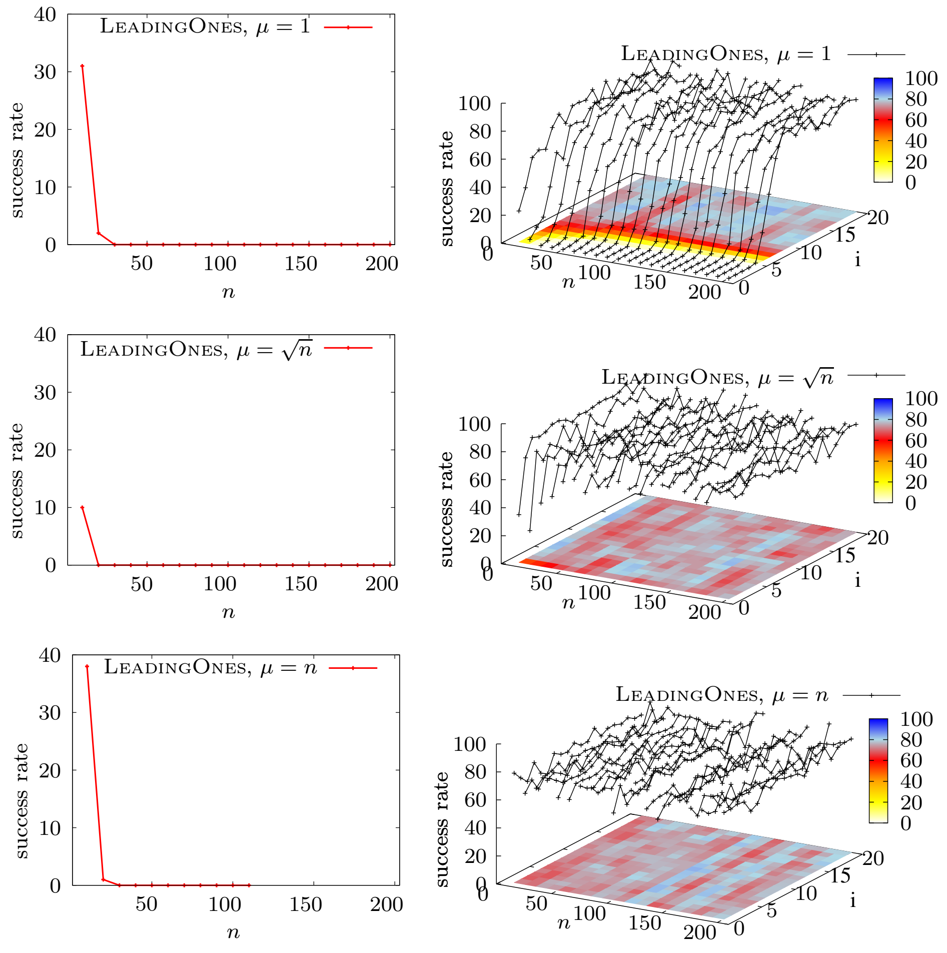


Figure 8.9.: Success rates in 100 independent runs of A_{eva} on LeadingOnes with $\tau_{\max} = 3$ (left) and $\tau_{\max} = (i/20) \cdot \lfloor 6 \log(n)\mu \log \mu \rfloor$ (right), and different sizes of the collection of search points.

9. Comparing Different Aging Strategies

In the previous chapter we have investigated the role of the most important parameter for aging operators, namely the maximal lifespan $\tau_{\max}$. Keeping these results in mind, we are now ready to compare different aging operators introduced in Chapter 7: on one hand static pure aging (Algorithm 7.4) from the field of artificial immune systems, on the other hand evolutionary aging (Algorithm 7.5) from the field of evolutionary computation.

An objective function f contributes decisively to a search landscape that can exhibit different features that make the task of optimization difficult. Different methods and tricks have been introduced to various search heuristics to overcome these and other difficulties, among them the concept of aging. Although many kinds of difficult features are known (Jansen 2001), we only concentrate on two common features, plateaus of constant function values and local optima. A rigorous analysis for these two typical difficult situations sheds light on similarities and differences of the considered aging operators. One important goal is, e. g., to hinder the current collection of search points from becoming too similar to each other. Preserving some degree of diversity is thought to be helpful in many situations including avoiding getting stuck in local optima and exploring plateaus efficiently.

We start with the analysis of local optima in Section 9.1 and consider plateaus afterwards in Section 9.2. Based on our findings, a third aging operator is introduced in Section 9.3 that provably shares the advantages of both aging mechanisms. Finally, experimental supplements are provided to point out practical implications of the theoretical results and discuss further issues concerning the considered aging strategies in Section 9.4. This chapter is based on the work done in Jansen and Zarges (2009b, 2011b).

9.1. Performance in Local Optima

Local optima are points in the search space where no neighbor has a strictly larger function value. They are a meaningful concept if the concrete notion of a neighborhood is connected to the search behavior of the randomized search heuristics. In the search space $\{0,1\}^n$ the Hamming neighborhood often has this property. Many randomized search heuristics consider search points with a small Hamming distance to a current search point with much higher probability than search points with larger Hamming distances. This is almost necessarily the case since the number of points with Hamming distance d increases exponentially in d (for not too large values of d). However, note that this is different for hypermutations discussed in Part II of this thesis.

It is a common experience that search heuristics get trapped in local optima. Realistic problems tend to be multi-modal and thus contain local optima. Therefore, considering example problems with well-defined local optima is of practical importance. The example function with a local optimum that we define follows the pattern of such example functions introduced by Jansen (2002). One example is a function called HSP_k that contains a broad and easy to find path to a local optimum and a narrow and hard to find path to the unique global optimum. The parameter k influences the likelihood of encountering the local optimum. Since this example function works only as desired for algorithms with very small μ we define a variant with very similar properties that also works for larger values of μ (Definition 9.1). Note, that this function is quite similar to the function f_k (Definition 8.10) and uses again the idea of two paths that are entered with different probabilities (see Section 8.2).

Definition 9.1. *For $n \in \mathbb{N}$ and $k \in \mathbb{N}$ with $k = O(1)$ the function* $\text{LocalOpt}_k\colon \{0,1\}^n \to \mathbb{R}$ *is defined by*

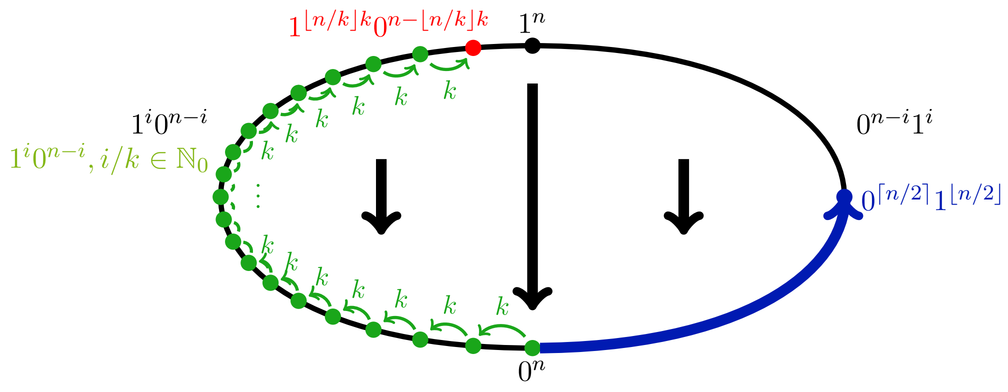
$$\text{LocalOpt}_k(x) = \begin{cases} n \cdot (i \cdot k + 1) & \textit{if } x \in \left\{1^{i \cdot k} 0^{n - i \cdot k} \mid i \in \{1, 2, \dots, \lfloor n/k \rfloor\}\right\}, \\ n \cdot (i + 1) & \textit{if } x \in \left\{0^{n-i} 1^i \mid i \in \{1, 2, \dots, \lfloor n/2 \rfloor\}\right\}, \\ n - \text{OneMax}(x) & \textit{otherwise.} \end{cases}$$

The example function LocalOpt_k is visualized in Figure 9.1. In the vast majority of the search space the fitness value of LocalOpt_k equals $n - \text{OneMax}(x)$ guiding a search heuristic towards the all-zero bit string 0^n. There, two different paths begin. One path is entered by changing i of the right-most bits to 1 (with $i \in \{1, 2, \dots, \lfloor n/2 \rfloor\}$). Increasing the number of 1-bits from right to left leads to the local optimum with function value $n \cdot (\lfloor n/2 \rfloor + 1)$. The other path is entered by changing exactly $i \cdot k$ of the left-most bits to 1 (with $i \in \{1, 2, \dots, \lfloor n/k \rfloor\}$). The function value can be increased on this path in this manner until the global optimum $1^{\lfloor n/k \rfloor \cdot k} 0^{n - \lfloor n/k \rfloor \cdot k}$ with function value $n \cdot (\lfloor n/k \rfloor \cdot k + 1)$ is found. The larger k is, the longer it takes to reach this global optimum and the more difficult it becomes to find the beginning of this path. Thus, we can expect randomized search heuristics to find the local optimum regularly for $k > 1$. For $k = 1$, the local and global path are entered with almost equal probability. Since the Hamming distance between the local optimum and the other path is $\Theta(n)$, any reasonable search heuristic should need an exponential number of steps to find the global optimum once the local optimum is found. For Algorithm 7.1 with $\tau_{\max} = \infty$ ((μ+1) EA) this is the case. We remark, that for LocalOpt_k, results for the (1+1) EA are known (Jansen 2002).

We consider the example function LocalOpt_k with sufficiently small parameter k and the performance of the two aging operators. To allow discussions on how aging can improve the performance of our algorithm, we first examine the performance of Algorithm 7.1 without aging by assuming that $\tau_{\max} = \infty$ holds and consider changes due to finite values for $\tau_{\max}$ afterwards. Recall, that this equals the (μ+1) EA (Definition 3.1).

Theorem 9.2 (Jansen and Zarges (2011b))**.** *Let $k \in \mathbb{N}$ with $k = O(1)$, $\tau_{\max} = \infty$, and $\mu \in \mathbb{N}$ arbitrarily. Then, for any number of steps $t \in \mathbb{N}$, the optimization time of the* (μ+1) EA *on* LocalOpt_k *is*

$$T_{(\mu+1)\text{ EA}, \text{LocalOpt}_k} \geq t$$



$1^{\lfloor n/k \rfloor k}0^{n-\lfloor n/k \rfloor k}$
1^n
1^i0^{n-i}
$1^i0^{n-i}, i/k \in \mathbb{N}_0$
$0^{n-i}1^i$
$0^{\lceil n/2 \rceil}1^{\lfloor n/2 \rfloor}$
k
0^n

Figure 9.1.: Visualization of the fitness function LocalOpt_k.

with probability

$$1 - O\left(\frac{\mu \log \mu}{n^{k-1}} + \frac{t}{n^{\frac{n}{2}-k}}\right).$$

For $\mu = o\left(n^{k-1}/\log n\right)$ *the expected optimization time is*

$$E\left(T_{(\mu+1)\text{ EA},\text{LocalOpt}_k}\right) = \Omega\left(n^{\frac{n}{2}-k}\right).$$

Proof. Note that for $\mu \log \mu = \Omega\left(n^{k-1}\right)$ and $t = \Omega\left(n^{(n/2)-k}\right)$ the statement becomes trivial. Thus, we assume $\mu \log \mu = o\left(n^{k-1}\right)$ in the following.

Assume that the complete collection of search points is on the path to the local optimum, i. e., of the form $0^{n-i}1^i$ for possibly different values of i with $i \geq 1$ for all of them. Then the global optimum can only be reached via a direct mutation to the other path. Such a mutation has probability at most $1/n^{i+\max\{i,k\}}$ since the i right-most 1-bits need to be changed into 0-bits and we need to reach a point on the other path that has at least the same function value. We consider μn^2 iterations. The probability not to make such a step in these iterations is bounded below by μ/n^{k-1}.

On the other hand, with probability $1 - e^{-\Omega(n)}$ in these steps the current best of the collection of search points is increased in function value. Then after on average $O(\mu \log \mu)$ iterations the function value of each member of the collection of search points is increased to at least this function value since it suffices to make copies of the current best (Witt 2006). The probability to create a search point on the other path in this time is bounded above by $O\left((\mu \log \mu)/n^{k+i}\right)$. Then we are in the same situation as before with $i \geq 2$. Summing up the probabilities to reach the other path for $1 \leq i \leq n/2$ we obtain $O\left((\mu \log \mu)/n^{k-1}\right)$ as bound. Thus, with probability $1 - O\left((\mu \log \mu)/n^{k-1}\right)$ the local optimum takes over the complete collection of search points. In this case a mutation of at least $(n/2) - k$ bits is necessary to reach the path. Such an event occurs in t steps with probability at most $O\left(t/n^{(n/2)-k}\right)$.

We still need to prove that the complete collection of search points gets on the path to the local optimum with sufficiently large probability. It is easy to see that each member of the collection of search points either reaches one of the two paths or 0^n within $O(\mu n^2)$ steps with probability $1-e^{-\Omega(n/\log n)}$ (Witt 2006). Consider the set of search points with exactly i 1-bits. For each $i \in \{k, k+1, \ldots, n\}$ there is at most one point on the path leading to the global maximum. As long as no member of the collection of search points is on any of the two paths each point in the search space with exactly i 1-bits has equal probability to become a member of the collection of search points. Thus, for each i the path to the global optimum is entered with probability at most $O(1/n^i)$. Summing up these probabilities for all $i \geq k$ we get the bound $O(1/n^k)$ on the probability to enter the path to the global optimum before either 0^n or the path to the local optimum is found. Consider some member of the collection of search points $x = 0^n$. Due to the strict selection employed we need only care about search points on one of the two paths. The probability to create some point on the path to the global optimum based on x given that a point on one of the two paths is created is $\Theta(1/n^{k-1})$. Thus, with probability $1 - O(1/n^{k-1})$ we get in the situation with the collection of search points on the path.

For $\mu = o(n^{k-1}/\log n)$ we get $\Omega(n^{(n/2)-k})$ as lower bound on the expected optimization time using the law of total probability (Lemma B.11). □

We have seen that for Algorithm 7.1 without aging the local optimum is encountered with probability close to 1 implying a very large expected optimization time if the size of the collection of search points is not too big. In the following we learn that evolutionary aging is not capable of improving the performance of our algorithm as it neither increases the probability of reaching the global optimum nor helps to escape the local optimum.

Theorem 9.3 (Jansen and Zarges (2011b)). *Let $k \in \mathbb{N}$ with $k = O(1)$, $\tau_{\max} \in \mathbb{N}$ arbitrarly, and $\mu \in \mathbb{N}$ arbitrarily. Then, for any number of steps $t \in \mathbb{N}$, the optimization time of the A_{eva} on* LocalOpt$_k$ *is*

$$T_{A_{eva},\text{LocalOpt}_k} \geq t$$

with probability

$$1 - O\left(\frac{\mu \log \mu}{n^{k-1}} + \frac{t}{n^{\frac{n}{2}-k}} + \frac{\mu t}{2^n \cdot \tau_{\max}}\right).$$

For $\mu = o(n^{k-1}/\log n)$ the expected optimization time is

$$E(T_{A_{eva},\text{LocalOpt}_k}) = \Omega(2^n).$$

Proof. As in the proof of Theorem 9.2 we assume $\mu \log \mu = o(n^{k-1})$ as otherwise the statement becomes trivial and now consider changes due to a finite value of $\tau_{\max}$.

If $\tau_{\max} = o(\mu n)$ holds, with probability $1 - e^{-\Omega(n)}$ not even one of the two paths is reached (Witt 2006). Thus, we assume $\tau_{\max} = \Omega(\mu n)$ in the following. Since evolutionary aging is employed, a new current member of the collection of search points starts with age 0. With probability at least $\Omega(1/\mu)$, the current best member of the collection of search points is copied. Thus, we manage to make a replica of the current best before it is removed due to its age with probability $1-e^{-\Omega(n)}$. Then nothing changes from the line of reasoning above since any new search point that is created in line 8 of Algorithm 7.1 will be replaced by a copy of the current best member of the collection of search points or a search point with even larger function value with sufficiently large probability before reaching the path to the global optimum.

Again we get $\Omega\big(n^{(n/2)-k}\big)$ as lower bound on the expected optimization time if $\tau_{\max}$ is sufficiently large. If $\tau_{\max}$ is very small almost constantly new search points are created uniformly at random. In t time steps, however, at most $t \cdot \mu/\tau_{\max}$ new search points are created in this way. Each of these new search point is equal to the unique global optimum with probability 2^{-n}. Moreover, this process finds the unique optimum on expectation in 2^n steps. This implies the desired probability and the lower bound on the expected optimization time. □

When using static pure aging instead of evolutionary aging at first sight not much is changed. If the maximal lifespan $\tau_{\max}$ is sufficiently large the collection of search points will gather in the local optimum with probability close to 1 (for not too large μ). There static pure aging makes a difference. Since no new search points with larger function values can be created unless the global optimum is found we only create new search points that inherit their age. Creating a copy of a current best search point is much faster than finding the local optimum. So, no new search points enter the local optimum. Thus, after some time, the complete collection of search points shares the very same age. This implies that at some point of time the complete collection of search points is replaced by new search points generated uniformly at random being equal to a restart. Since the path to the global optimum is found with not too small probability we expect to find the global optimum after not too many restarts.

Theorem 9.4 (Jansen and Zarges (2011b))**.** *Let $k \in \mathbb{N}$ with $k = O(1)$, $\mu \in \mathbb{N}$ arbitrarly, and $\tau_{\max} \in \mathbb{N}$ with $\tau_{\max} = \omega\big(\log(n) \cdot (n^k + \mu \log n)\big)$. The expected optimization time of A_{spa} on* LocalOpt$_k$ *is*

$$E\left(T_{A_{spa},\textsc{LocalOpt}_k}\right) = O\Big(\tau_{\max} \cdot n^{k-1} + n^{k+1}\Big).$$

Proof. Using insights from previous proofs and results on the $(\mu{+}1)$ EA due to Witt (2006) the proof is relatively simple. Given that the local optimum is found the expected number of steps to do so is bounded by $O\big(n^2 + \mu n \log n\big)$. In the same way we see that given that the global optimum is found the expected number of steps to do so is bounded by $O\big(n^{k+1} + \mu n \log n\big)$. Since $\tau_{\max} = \omega\big(\log(n) \cdot (n^k + \mu \log n)\big)$ this bound carries over to A_{spa} (Theorem 8.1 and 8.11).

From the proof of Theorem 9.2 we know that we enter the path to the global optimum with probability $\Omega\big(1/n^{k-1}\big)$. Thus, on average after $O\big(n^{k-1}\big)$ 'restarts' of the algorithm this happens. We have such a 'restart' if all search points are removed due to their age simultaneously. This happens in the local optimum after each search point was created as a copy of the current best search point. The expected time for this to happen is $O(\tau_{\max} + \mu \log \mu)$ since the time to have the collection of search points taken over by the youngest current best is $O(\mu \log \mu)$ (Witt 2006) and after $O(\tau_{\max})$ iterations all older current bests are removed due to their age. Moreover, the expected time to reach the local optimum is bounded by $O\big(n^2 + \mu n \log n\big)$. Together this yields $O\big(\tau_{\max} + n^2 + \mu n \log n\big)$ as upper bound on the expected waiting time for a 'restart.' Multiplying it with the expected number of 'restarts' $O\big(n^{k-1}\big)$ we obtain $O\big(\tau_{\max} \cdot n^{k-1} + n^{k+1} + \mu n^k \log n\big)$. as bound for this part. Since this dominates the upper bound for the expected time to reach the global optimum once the path leading to it is found we have this bound as upper bound on the expected optimization time. With $\tau_{\max} = \omega\big(\log(n) \cdot (n^k + \mu n)\big)$ the bound simplifies to $O\big(\tau_{\max} \cdot n^{k-1} + n^{k+1}\big)$ since $\tau_{\max} \cdot n^{k-1} = \Omega\big(\mu n^k \log n\big)$ holds. □

We have seen that static pure aging can increase the performance of a randomized search heuristics dramatically by allowing it to perform restarts. Doing so static pure aging enables the algorithm to escape from local optima yielding a polynomial expected optimization time whereas the algorithm without aging and using evolutionary aging respectively gets trapped in the local optima with probability close to 1 implying a very large expected optimization time if the size of the collection of search points is not too big. We present results of experiments in order to point out practical implications of our theoretical results and discuss further issues concerning the considered aging strategies in Section 9.4.1.

9.2. Performance on Plateaus

A plateau is a set of neighboring points in the search space with equal function value. Again, we assume that Hamming distance is appropriate and consider two points to be neighbors if their Hamming distance equals 1. Plateaus turn out to be obstacles if they are not very small since they give no hint at all in what direction to search. Therefore, a kind of random walk on the plateau has to be performed that can be quite time consuming. In practical applications, in particular in combinatorial optimization problems, it is often the case that the set of potential solutions is exponentially large whereas the set of solution values is only polynomial in size. This makes it likely that plateaus exist making their consideration in example problems practically relevant.

We consider an example function, containing a plateau of n points spanning a large Hamming distance of $n-1$ between the first and the last point on the plateau, introduced by Jansen and Wegener (2002). We call it Plateau and define it as follows.

Definition 9.5 (Jansen and Wegener (2002)). *For $n \in \mathbb{N}$, the function* Plateau: $\{0,1\}^n \to \mathbb{R}$ *is defined for by*

$$\textsc{Plateau}(x) = \begin{cases} n & \text{if } x \in \big\{1^i 0^{n-i} \mid i \in \{0,1,2,\dots,n-1\}\big\}, \\ n+1 & \text{if } x = 1^n, \\ n - \textsc{OneMax}(x) & \text{otherwise.} \end{cases}$$

Like for $\textsc{LocalOpt}_k$, in the vast majority of the search space the fitness values guide a search heuristic towards 0^n. There, the plateau of points $1^i 0^{n-i}$ begins. Since there are

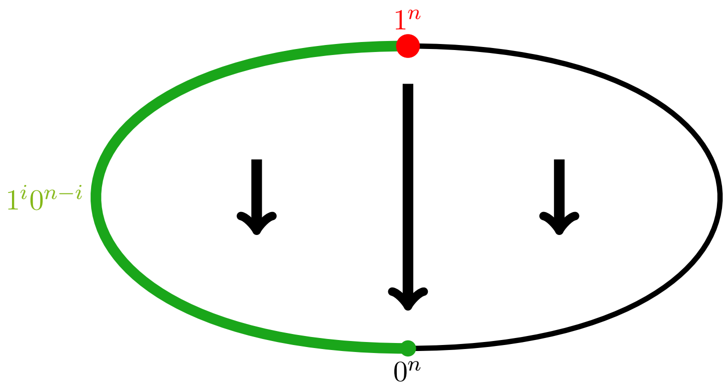


Figure 9.2.: Visualization of the fitness function PLATEAU.

no hints in which direction better search points can be found and since all points not on the plateau have worse function value, usually randomized search heuristics will perform a random walk on the plateau until they happen to find the unique global optimum 1^n. An illustration of PLATEAU can be found in Figure 9.2. For PLATEAU results for the (1+1) EA (Jansen and Wegener 2002), the (μ+1) EA (Witt 2006), and some similar evolutionary algorithms without aging (Friedrich et al. 2009a) are known.

The proofs in this section again use the method of family trees introduced by Witt (2006) and described earlier in Section 4.3.2 just before Theorem 4.21. Recall, that a family tree $T_t(x_0)$ contains the direct and indirect descendants of a search point x_0 created by time $t \geq 0$ via mutation where nodes of the tree identify the search points generated and an edge (x, y) denotes that y was created by mutating x. The depth of a family tree is defined as the maximal depth of its nodes. Note that due to the use of aging there can exist more than μ family trees as each new search point that is created in line 8 of Algorithm 7.1 generates a new one. When showing upper bounds on the optimization time using lower bounds on the depth of the family tree we need to take care that not too many additional family trees are created.

A tree $T_t(x_0)$ can contain search points that have already been deleted from the collection of search points at time t. It can even happen that $T_t(x_0)$ only contains deleted search points. It is obvious that at least one family tree in the collection must contain at least one search point that has not been deleted yet. Considering a node $y \in T_t(x_0)$ and a path P from x_0 to y we say that y is *alive* if $y \in C_t$ and *dead* otherwise. A path P is *alive* if it contains at least one alive node. There is always at least one alive path in some family tree. Analogously to Witt (2006) we call search points on the plateau, i. e. points of the shape 1^i0^{n-i} for $0 \leq i \leq n$, *plateau points*. Moreover, we denote the first point in time where all search points are plateau points $T'_{\text{PLATEAU}} = \min\left\{t \mid C_t \subseteq \{1^i0^{n-i} \mid 0 \leq i \leq n\}\right\}$. A figure containing example family trees for PLATEAU with $n = 6$ and $\mu = 2$ for a small number of iterations is given in Figure 9.3. Note that the ordering of removals of search points from the collection of search points is not completely defined by the offspring but contains a random element.

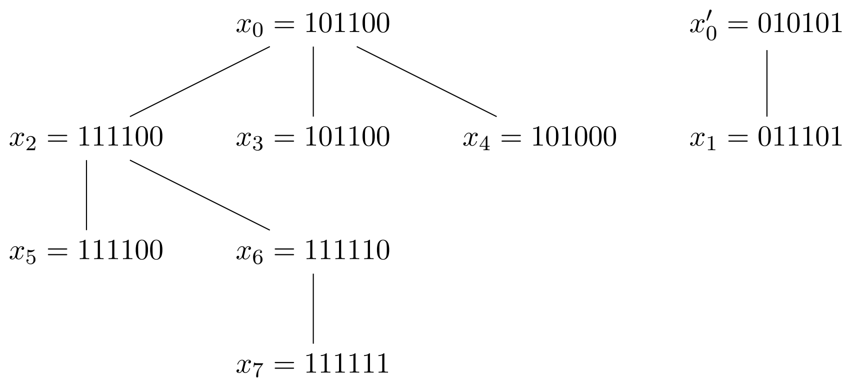


Figure 9.3.: Two family trees for Plateau with $n = 6$ and $\mu = 2$. The search point x_i is produced in the i-th iteration of the algorithm. The search points x_0 and x'_0 are the two search points from the initial collection of search points. The search points are removed from the current collection of search points in the following ordering: x_1, x'_0, x_3, x_0, x_4, x_2, x_5. Already in iteration 2 the family tree with root x'_0 contains only dead nodes. The optimization time equals $T = 7$. We have $T'_{\text{Plateau}} = 5$.

As for LocalOpt$_k$ we compare the performance of the (μ+1) EA, A_{spa}, and A_{eva} in order to discuss the effects of the considered aging strategies. For Plateau the (μ+1) EA was already analyzed by Witt (2006). For the sake of completeness we simply restate the corresponding theorem here.

Theorem 9.6 (Theorem 3 from Witt (2006))**.** *Let $\mu = n^{O(1)}$. The expected optimization time of the* (μ+1) EA *on* Plateau *is*

$$E\left(T_{(\mu+1)\text{ EA},\text{Plateau}}\right) = O\left(\mu n^3\right).$$

The main idea of the proof of Theorem 9.6 is to rediscover a run of the (1+1) EA, i. e., of a single search point, on Plateau on alive paths of a family tree (Witt 2006). It is known that the expected optimization time of the (1+1) EA on Plateau is $O\left(n^3\right)$ (Droste et al. 2002) which implies a lower bound on the expected depth of such a family tree. One can conclude that the expected optimization time of the (μ+1) EA is bounded above by the sum of $\mathrm{E}\left(T'_{\text{Plateau}}\right)$ and the expected time until the family tree reaches depth $\Theta\left(n^3\right)$.

We first consider evolutionary aging and prove asymptotically the same bound on the optimization time as for the (μ+1) EA without making strong assumptions on the maximal lifespan $\tau_{\max}$. The proof follows the line of thought of Witt (2006) for the (μ+1) EA. Therefore, we concentrate on modifications needed due to the use of aging.

Theorem 9.7 (Jansen and Zarges (2011b))**.** *Let $\tau_{\max} = \omega(\log(n) \cdot (n + \mu \log n))$ and $\mu = n^{O(1)}$. The expected optimization of A_{eva} on* Plateau *is*

$$E(T_{A_{eva},\text{Plateau}}) = O\left(\mu n^3\right).$$

Proof. From the proof for the $(\mu{+}1)$ EA we know that $\mathrm{E}\left(T'_{\textsc{Plateau}}\right) = O(\mu n + n \log n) = O(\mu n \log n)$ holds (Witt 2006). As we use aging it remains to show that it is unlikely that any currently best search point is removed due to its age before creating an offspring with strictly larger function value in order to obtain the same bound on $\mathrm{E}\left(T'_{\textsc{Plateau}}\right)$. Afterwards we prove that the expected time until the family tree reaches some depth k can be bounded above by $O(\mu k)$ once all search points in the collection of search points are plateau points. This implies an upper bound on the optimization time of $O\left(\mu n^3\right)$ since due to the results for the (1+1) EA (Droste et al. 2002) the optimum is obtained after an expected path length of $O\left(n^3\right)$ in a family tree.

Since for $T'_{\textsc{Plateau}}$ we essentially consider OneMax, the maximal lifespan $\tau_{\max}$ is sufficiently large to carry over the upper bound on $\mathrm{E}\left(T'_{\textsc{Plateau}}\right)$ (Theorem 8.4 and Corollary 8.5). Once a plateau point has been created, the number of those points is increased if we choose a plateau point and produce a replica. The expected time until the collection of search points only consists of copies of this point is again $O(\mu \log \mu) = O(\mu \log n)$ (Witt 2006). Clearly, our lower bound on $\tau_{\max}$ is large enough for moving all points on the plateau.

The upper bound on the expected time to reach depth k in a family tree remains to be shown. Let S_t be the set of alive search points in $T_t(x_0)$ that always have an alive descendant until time $T'_{\textsc{Plateau}} + 4e\mu k$. Let L_t denote the maximum depth of $x \in S_t$ in $T_t(x_0)$. L_t increases if a search point x with $\mathrm{depth}\,(x) = L_t$ is selected, a plateau point x' is created and x is deleted before x'. The probability is $1/\mu$ for the first event and $(1 - 1/n)^n \geq 1/(2e)$ for the second event since it is sufficient to produce a replica of x.

For the third event again aging comes into play. The probability that x is deleted before x' is $1/2$ due to selection as both search points have the same function value. It is not possible that x' is deleted before x due to its age as we use evolutionary aging. It remains to show that x' is generated with high probability before x is deleted due to its age. We show that with probability $1 - n^{-\omega(\log n)}$ after $T'_{\textsc{Plateau}}$ no search point is deleted due to its age.

In each iteration after $T'_{\textsc{Plateau}}$ a new plateau point is generated with probability $\Omega(1)$ since creating a replica of any parent is sufficient. Thus, with probability $1 - e^{-\omega\left(\mu \log^2 n\right)}$ in $\omega\left(\mu \log^2 n\right)$ iterations we have $\omega\left(\mu \log^2 n\right)$ new plateau points by application of Chernoff bounds. Consider some member of the collection of search points. It is not removed due to selection in a single iteration with probability $1 - 1/(\mu + 1)$. Thus, with probability $(1 - 1/(\mu + 1))^{\omega\left(\mu \log^2 n\right)} = n^{-\omega(\log n)}$, it survives the production of $\omega\left(\mu \log^2 n\right)$ new plateau points. Note that we have $\tau_{\max} = \omega\left(\mu \log^2 n\right)$. Thus, with probability $1 - \mu \cdot n^{-\omega(\log n)} = 1 - n^{-\omega(\log n)}$ no member of the collection of search points reaches its maximal lifespan $\tau_{\max}$. Thus, with this probability aging does not come into play at all. This yields an expected number of $4e\mu k = O(\mu k)$ steps for reaching depth k.

As we have seen after $T'_{\textsc{Plateau}}$ with probability $1 - n^{-\omega(\log n)}$ no new random search points are needed to replace search points that are removed due to their age. Thus, with probability $1 - n^{-\omega(\log n)}$ no new family trees are created. $\square$

We have seen that using evolutionary aging with sufficiently large maximal lifespan $\tau_{\max}$ does not change much compared to the corresponding algorithm without aging on PLATEAU. This is not the case when static pure aging is used as descendants of plateau points that are plateau points themselves inherit the age of the parent. This may lead to the extinction of the whole collection of search points on the plateau if the maximal lifespan $\tau_{\max}$ is not sufficiently large. We formalize this in the next theorem.

Theorem 9.8 (Jansen and Zarges (2011b))**.** *Let $\mu = n^{O(1)}$, $\alpha(n) = \omega(1)$ and $\alpha(n) = O(n/\ln n)$, and $\tau_{\max} = \omega(\log(n) \cdot (n + \mu \log n))$ and $\tau_{\max} = O\big(n^3/(\alpha(n)^{3/2} \ln n)\big)$. The optimization time of A_{spa} on* PLATEAU *is*

$$T_{A_{spa},\text{PLATEAU}} = n^{\Omega\left(\sqrt{\alpha(n)}\right)}$$

with probability $1 - n^{-\Omega\left(\sqrt{\alpha(n)}\right)}$.

Proof. From the proof of Theorem 9.7 we know that $\mathrm{E}\left(T'_{\text{PLATEAU}}\right) = O(\mu n \log n)$ holds. As the lower bound on $\tau_{\max}$ here matches that of Theorem 9.7, we can assume that all search points in the collection of search points are plateau points. All collections of search points until (and including) time T'_{PLATEAU} only contain search points with at most $3n/5$ 1-bits with probability $1 - 2^{-\Omega(\sqrt{n})}$, implying that all search points so far have linear Hamming distance to the optimum. This is Lemma 3 in Witt (2006).

We now show that the collection of search points becomes extinct on the plateau with probability $1 - n^{-\Omega\left(\sqrt{\alpha(n)}\right)}$ after at most $\tau_{\max} = O\big(n^3/(\alpha(n)^{3/2} \ln n)\big)$ steps. Assume that $x \in C_t$, $t > T'_{\text{PLATEAU}}$, is selected. We denote a step as *relevant* if and only if it generates a plateau point $y \neq x$. Recall that y inherits the age of x in this case. We bound the number of relevant steps within overall $\tau_{\max}$ steps and the step size of such a relevant step from above. Finally we show that this leads with high probability to the extinction of the collection of search points before the optimum is reached and thus, to a super-polynomial lower bound on the optimization time.

We first consider only a single search point in the collection of search points. The probability for having a relevant step is at most $2/n$ as either the leftmost 0-bit or the rightmost 1-bit has to flip. Thus, in a phase of $O\big(n^3/(\alpha(n)^{3/2} \ln n)\big)$ steps there are at most $O\big(n^2/(\alpha(n)^{3/2} \ln n)\big)$ relevant steps with probability $1 - 2^{-\Omega\left(n^2/(\alpha(n)^{3/2} \ln n)\right)}$ using Chernoff bounds. Here, the upper bound on $\alpha(n)$ is needed.

A relevant step with step size b requires at least a b-bit mutation which happens with probability at most $2/n^b$. The probability of not having a relevant step with step size at least $3 + \sqrt{\alpha(n)}$ in $\tau_{\max} = O\big(n^3/(\alpha(n)^{3/2} \ln n)\big)$ steps is then bounded below by $1 - \tau_{\max} \cdot O\left(n^{-(3+\sqrt{\alpha(n)})}\right) = 1 - n^{-\Omega\left(\sqrt{\alpha(n)}\right)}$.

We assume pessimistically that all relevant steps have step size $3 + \sqrt{\alpha(n)}$ and that the last such step reaches the optimum. The resulting random process corresponds to a fair random walk on at most

$$m := \frac{(2n/5) - 3 - \sqrt{\alpha(n)}}{3 + \sqrt{\alpha(n)}}$$

states. We want to bound from above the probability to overcome the distance m, i.e., finding the optimum, in $s \leq n^2/(4\alpha(n)^{3/2} \ln n)$, steps. As we have a fair random walk, the probability for steps in either direction (say *left* and *right*) is 1/2 and the expected number of steps for each direction in s steps equals $s/2$. We use Chernoff bounds to bound the probability sought in the following way:

$$\begin{aligned}\text{Prob}\,(\text{in } s \text{ steps distance } \geq m) &\leq \text{Prob}\left(\text{in } s \text{ steps } \leq \frac{s}{2} - \frac{m}{2} \text{ times left}\right)\\ &= \text{Prob}\left(\text{in } s \text{ steps } \leq \frac{s}{2} \cdot \left(1 - \frac{m}{s}\right) \text{ times left}\right) \leq e^{-\frac{s}{2}\cdot\frac{m^2}{s^2}\cdot\frac{1}{2}} = e^{-\frac{m^2}{4s}}\end{aligned}$$

Plugging $s = n^2/(4\alpha(n)^{3/2} \ln n)$ and $m = cn/\sqrt{\alpha(n)}$ for some appropriate constant $0 < c < 2/5$ into this result yields

$$e^{-\frac{m^2}{4s}} = e^{-\frac{c^2 \cdot n^2 \cdot 4 \cdot \alpha(n)^{3/2} \cdot \ln n}{\alpha(n) \cdot n^2}} = n^{-\Omega\left(\sqrt{\alpha(n)}\right)}.$$

Using the simple union bound and that this probability is monotonically increasing in s, the probability to find the optimum within $\leq s$ steps is bounded by $s \cdot n^{-\Omega\left(\sqrt{\alpha(n)}\right)} = n^{-\Omega\left(\sqrt{\alpha(n)}\right)}$. Analogously, the probability that any search point in the collection of search points reaches the optimum in $O\big(n^3/(\alpha(n)^{3/2} \log n)\big)$ steps is bounded by $\mu \cdot n^{-\Omega\left(\sqrt{\alpha(n)}\right)} = n^{-\Omega\left(\sqrt{\alpha(n)}\right)}$. □

We have seen that in the case of plateaus evolutionary aging neither improves nor worsens the performance of the randomized search heuristic in comparison to not using aging at all if the maximal lifespan $\tau_{\max}$ is sufficiently large. In contrast to that, static pure aging may hinder the search heuristic to perform a random walk on a plateau implying a large optimization time. To be more precise we have shown that for $\tau_{\max} = \omega(\log(n) \cdot (n + \mu \log n))$ and $\tau_{\max} = o\big(n^3/(\alpha(n)^{3/2} \ln n)\big)$ for some $\alpha(n) = \omega(1)$ and $\alpha(n) = O(n/\ln n)$ evolutionary aging yields the same upper bound on the optimization time as the algorithm without aging whereas the algorithm with static pure aging has superpolynomial optimization time. If $\tau_{\max}$ is chosen significantly larger than that also static pure aging is efficient. This is not surprising, since the (μ+1) EA, i.e., Algorithm 7.1 with $\tau_{\max} = \infty$, has polynomial optimization time. However, by increasing the maximal lifespan $\tau_{\max}$ aging becomes more and more ineffective as the frequency of reaching the maximal lifespan decreases and, thus, aging achieves nothing in practice.

If the maximal lifespan $\tau_{\max}$ is chosen significantly smaller both aging mechanisms are inefficient. Note that the plateau embedded in our example function only has size n. On bigger plateaus static pure aging has even bigger difficulties. Again we provide experimental supplements to point out practical implications of our theoretical results and discuss further issues concerning the considered aging strategies in Section 9.4.2 .

9.3. Combining the Assets of Aging

We have seen that static pure aging is able to efficiently optimize functions with local optima where evolutionary aging fails. This is due to the capability of performing restarts inherent to static pure aging. On the other hand we have seen that static pure aging with a maximal lifespan $\tau_{\max}$ that is not very large fails on plateaus where evolutionary aging has no problems. This failure is caused by incompetency of static pure aging in recognizing that it is making some kind of progress even though the function values of the search points encountered do not improve. While recognizing stagnation by measuring function values helps to escape from local optima, this very mechanism hinders static pure aging to be efficient on even rather small plateaus. When we consider both situations in direct comparison it is not difficult to spot a crucial difference. While the function values do not increase in both situations being stuck in a local optimum means that also no new search points with equal function values are discovered. When a random walk on the plateau is performed, there are constantly new points discovered even though they all have equal function value. We introduce an aging operator called genotypic aging that spots this difference (Definition 9.1). Obviously, we denote Algorithm 7.1 using genotypic aging by A_{ga}.

Algorithm 9.1 Genotypic Aging (ga).

```
1. If $f(y) \geq f(x)$  and  $y \neq x$ then
2.     Set $y$.age := 0.
3. Else
4.     Set $y$.age := $x$.age.
```

In comparison to static pure aging we changed the condition '$f(y) > f(x)$' to '$(f(y) \geq f(x)) \wedge (y \neq x)$'. Since $f(y) > f(x)$ implies $y \neq x$, there is no change for this case. If $f(y) = f(x)$ holds we make a case distinction based on x and y. Those are called genotypes in the context of evolutionary algorithms motivating our choice of genotypic aging as name for this operator. Finding a new search point with equal function value is now sufficient progress to warrant setting its age to 0. This allows for random walks on plateaus beyond what the maximal lifespan $\tau_{\max}$ allows.

We first consider LocalOpt$_k$ and show that restarts are still possible when genotypic aging is used. More precisely, genotypic aging behaves essentially the same as static pure aging on LocalOpt$_k$.

Theorem 9.9 (Jansen and Zarges (2011b)). *Let $k \in \mathbb{N}$ with $k = O(1)$, $\mu \in \mathbb{N}$ arbitrarily, and $\tau_{\max} \in \mathbb{N}$ with $\tau_{\max} = \omega\big(\log(n) \cdot (n^k + \mu \log n)\big)$. The expected optimization time of A_{ga} on* LocalOpt$_k$ *is*

$$E\left(T_{A_{ga},\text{LocalOpt}_k}\right) = O\Big(\tau_{\max} \cdot n^{k-1} + n^{k+1}\Big).$$

Proof. We can mostly re-use ideas from the proof of Theorem 9.4 as the two aging mechanisms genotypic aging and static pure aging behave very similarly on LocalOpt$_k$.

Nothing changes about the way the local optimum is found. This holds since the local optimum can be reached by a number of fitness improvements and the maximal lifespan $\tau_{\max}$ is sufficiently large to allow for each of them with a probability close to 1. Clearly, nothing changes about the probabilities to enter the two paths as these probabilities are determined by the variation operator, not by aging. The only thing we need to care about is if we still have 'restarts' when all search points are stuck in the local optimum. Note that the local optimum is a unique point $0^{\lfloor n/2 \rfloor}1^{n-\lfloor n/2 \rfloor}$ and that all points with equal or larger function value have Hamming distance $\Omega(n)$ to this point. Thus, with probability exponentially close to 1 no such search point will be encountered if $\tau_{\max} = n^{o(n)}$ holds. For even larger $\tau_{\max}$ the path to the global optimum may be discovered. Since this can only decrease the optimization time it does not hurt our upper bound. $\square$

In a similar way we can show that genotypic aging behaves similar to evolutionary aging on PLATEAU. In contrast to the proof of Theorem 9.7 we now have to keep in mind that replicas of a plateau point inherit the age of its parent. Therefore, the maximal lifespan $\tau_{\max}$ has to be a bit larger as now specific 1-bit mutations – instead of creating replicas – are required to obtain search points with age 0.

Theorem 9.10 (Jansen and Zarges (2011b)). *Let $\mu = n^{O(1)}$ and $\tau_{\max} = \omega(\mu n \log n)$. The expected optimization of A_{ga} on* PLATEAU *is*

$$E\left(T_{A_{ga},\text{PLATEAU}}\right) = O\left(\mu n^3\right).$$

Proof. Again we can re-use the ideas from a previous proof (Theorem 9.7) as on PLATEAU the aging mechanisms genotypic aging and evolutionary aging behave very similar. For the upper bound on T'_{PLATEAU} we have to consider the expected time for increasing the currently best function value. As in the proof of Theorem 9.7 we see that $\tau_{\max} = \omega(\mu n \log n)$ suffices to reach the plateau with the whole collection of search points with probability at least $1 - n^{-\omega(1)}$.

For the upper bound on the time to reach depth k in a family tree we have to be more careful as now replicas inherit the age of its parent. However, as the age of the replica and the parent are the same, the descendant can only be deleted before its parent due to selection which has still probability 1/2. Therefore, we only need to care about the probability to generate a descendant x' of a plateau point x with maximal depth L_t and consider relevant steps where $x \neq x'$. The probability for a relevant step is at least $1/n \cdot (1 - 1/n)^{n-1} \geq 1/(en)$ as it suffices to flip exactly one bit and thus, the expected waiting time for this is bounded by $O(n)$.

Similarly to evolutionary aging a search point created during a relevant step does not have larger fitness and therefore has equal probability to be removed during the fitness-based selection process. We see that it is not sufficient to consider a single relevant step but rather consider the expected time until all search points have been involved into a relevant step. The expected time for this event is bounded by $O(\mu n \log \mu)$ due to coupon collector arguments (Lemma B.17). Note that $\tau_{\max} = \omega(\mu n \log n)$ is large enough to wait for this and thus, the desired probability of $1 - n^{-\omega(1)}$ follows. $\square$

So far we have seen that aging can both increase and decrease the performance of a randomized search heuristic dramatically. On one hand aging allows to perform restarts and thus escape local optima. On the other hand aging may hinder a randomized search heuristic to perform a random walk on a plateau. The aging operator introduced here combines the benefits of both previous operators. We also provide experimental supplements for this operator in order to allow for comparisons in the next section.

9.4. Experimental Supplements

The results presented in the preceding sections give new insights into drawbacks and benefits of aging strategies for randomized search heuristics. Moreover, they give a coarse picture of what can happen during the optimization process when using different aging strategies. Nevertheless, not all questions are answered. First of all, we have only shown upper bounds on the expected optimization time in cases where aging is beneficial and upper bounds on success probabilities where it is harmful. Second, asymptotic results may not describe the situation for typical problem dimensions. Furthermore, the influence of the maximal lifespan $\tau_{\max}$ and the size μ of the collection of search points was not analyzed in detail. Therefore, in this section, we investigate further properties of our algorithm in order to supplement our theoretical results.

This experimental analysis here is similar to that executed in the preceding chapter, in particular with respect to the investigation of the influence of $\tau_{\max}$. Therefore, we begin with the same experimental setup as described previously.

First, we consider the situations which yield polynomial upper bounds on the optimization time, i. e., static pure aging and genotypic aging on $\textsc{LocalOpt}_k$ and evolutionary aging and genotypic aging on $\textsc{Plateau}$ respectively. We again use $\mu \in \{1, \lfloor\sqrt{n}\rfloor, n\}$ and values for $\tau_{\max}$ which are bounded below by our results. We perform 100 independent runs of the considered algorithm and plot the results using box-and-whisker plots (Definition B.1) for $n \in \{10, 20, \ldots, 200\}$.

Second, we analyze the influence of the maximal lifespan $\tau_{\max}$ for $n \in \{10, 20, \ldots, 200\}$ and the same settings of μ as in the first set of experiments in very much the same way as done before. However, we only consider $n \in \{10, 20, \ldots, 150\}$ for $\textsc{Plateau}$ and $\mu = n$ due to the excessive computation time. For each value of n and μ we perform experiments with 20 equidistantly chosen values for $\tau_{\max}$, i. e., $\tau_{\max} = (i/20) \cdot \tau_{\max}$ $(i \in \{1, \ldots, 20\})$ where $\tau_{\max}$ is the corresponding $\tau_{\max}$ value from the first set of experiments. We fix the maximal number of iterations executed to the corresponding upper quartile from the first set of experiments and give results in form of success rates as 3D plots.

In addition to the experimental analysis in the previous chapter, we perform a third set of experiments, which is concerned with the algorithms where aging is provably harmful, i. e., evolutionary aging on $\textsc{LocalOpt}_k$ and static pure aging on $\textsc{Plateau}$. Again we perform experiments for $\mu \in \{1, \lfloor\sqrt{n}\rfloor, n\}$ and $n \in \{10, 20, \ldots, 200\}$. We fix the maximal number of iterations executed to the corresponding (maximal) upper quartile from the first set of experiments. The results are given as a plot for each μ showing the number of successful runs within 100 independent runs for all pairs of n. As maximal lifespan $\tau_{\max}$

we use the values from the first set of experiments. Note that for PLATEAU we have different bounds for $\tau_{\max}$ in evolutionary aging and genotypic aging. Thus, there are different bounds in the experiments for evolutionary aging and genotypic aging.

9.4.1. Results for Local Optima

As described in Section 9.1 the idea behind LOCALOPT_k is that it contains an easy to find path to a local optimum and a hard to find path to the unique global optimum. Although LOCALOPT_k is defined for $k = 1$, this choice of k does not reflect the underlying idea of the function. Furthermore, the optimization times on LOCALOPT_k considered here are exponential in k. Therefore, we restrict our experiments to $k = 2$.

For the first set of experiments we consider static pure aging and genotypic aging. Due to Theorem 9.4 and Theorem 9.9 an appropriate maximal lifespan here is $\tau_{\max} = \omega\big(\log(n) \cdot (n^k + \mu n)\big)$ and the optimization time is $O\big(\tau_{\max} n^{k-1} + n^{k+1}\big)$. Thus, we set $\tau_{\max} = \big\lfloor 6\log^2(n) \cdot (n^k + \mu n)\big\rfloor$ and get $O\big(n^3 \log^2 n\big)$ for $k = 2$ and $\mu \in \{1, \lfloor\sqrt{n}\rfloor, n\}$.

The empirical results are shown in Figure 9.4. As before we plot $c \cdot b(n)$ if we have a bound of $O(b(n))$ on the expected optimization time for illustrative purposes. The constant c is determined using a least squares fit. It is evident that the three values of μ chosen here do not only lead to the same asymptotic upper bound on the optimization time but in fact exhibit a very similar behavior in our study as the number of iterations needed seems to be mostly independent of μ if μ is not too big. Furthermore the choice between static pure aging and genotypic aging does not yield any significant differences. We conclude that the optimization time is dominated by the number of restarts and the waiting time for a restart rather than the possible additional costs caused by the collection of search points or the actual choice between these two aging mechanisms.

The results of the second set of experiments are shown in Figure 9.5. Note that we expect 75 successful runs within 100 runs as we fix the maximal number of iterations executed to the corresponding upper quartile from the first set of experiments. Again both aging mechanism as well as the three settings for μ yield very similar results. We see that the algorithms seem to be rather robust against changes of the maximal lifespan $\tau_{\max}$. For most of the considered values the success rate is not below our expected value and only for the smallest three to five values it drops drastically. Note that the success rate becomes even larger if $\tau_{\max}$ is set to slightly smaller values as the ones that stem from our theoretical results. This suggests that we are conservative with respect to our choice of $\tau_{\max}$ and in practice slightly smaller values are to be preferred. For larger sizes of the collection of search points this effect becomes more obvious as the sudden decrease of the success rate appears later. We conclude that a bigger collection of search points might lead to more robustness against changes to the maximal lifespan $\tau_{\max}$. Moreover, the results indicate that the success rate seems to converge to 0 while $\tau_{\max}$ decreases. This leads to the conclusion that setting $\tau_{\max}$ to constant values is not appropriate. Note that for constant $\tau_{\max}$ almost constantly new search points are created uniformly at random, making the search heuristics very similar to pure random search which finds a unique global optimum in expected time 2^n.

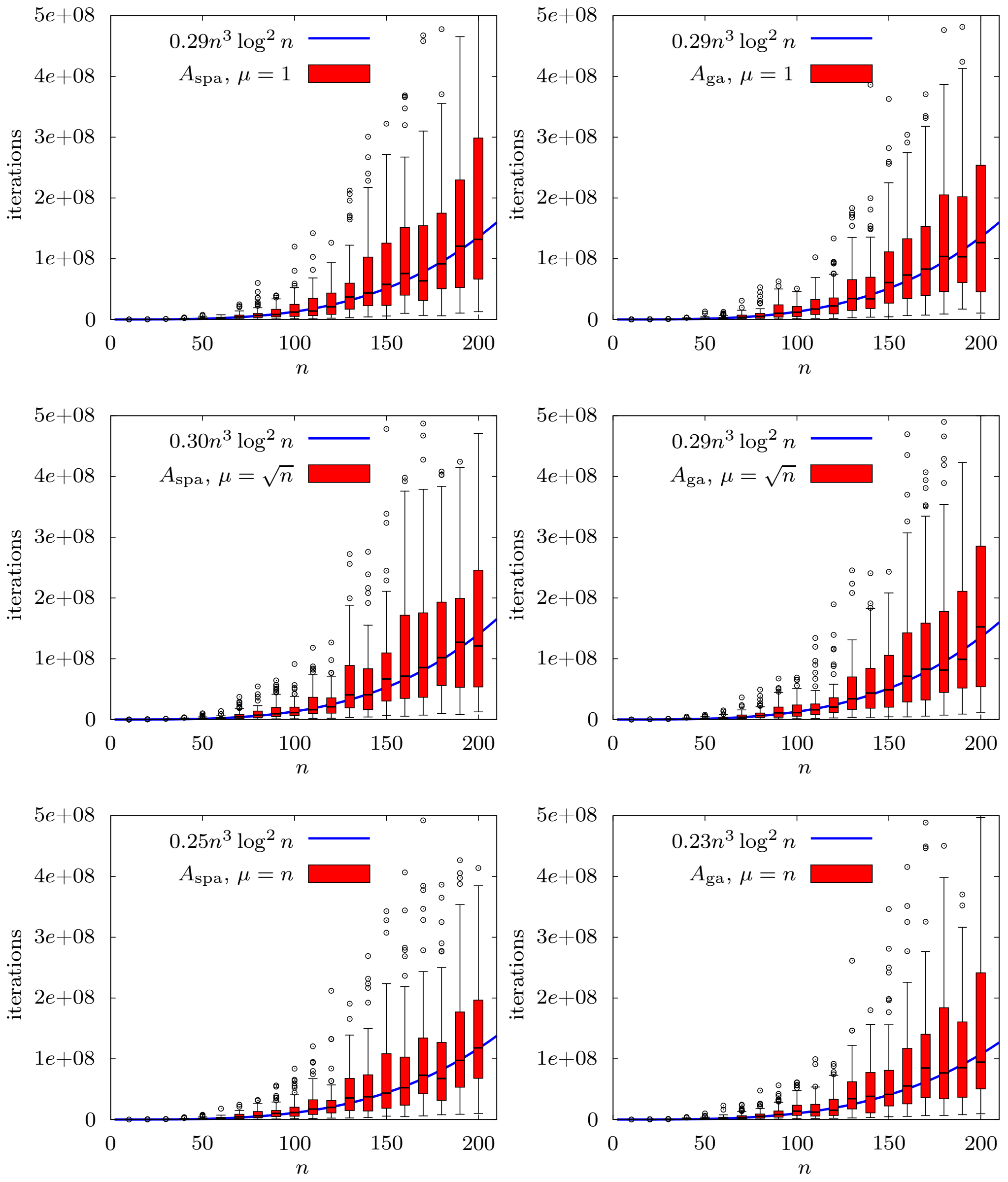


Figure 9.4.: Empirical results for A_{spa} and A_{ga} on LocalOpt$_k$ with different sizes μ of the collection of search points, data from 100 runs.

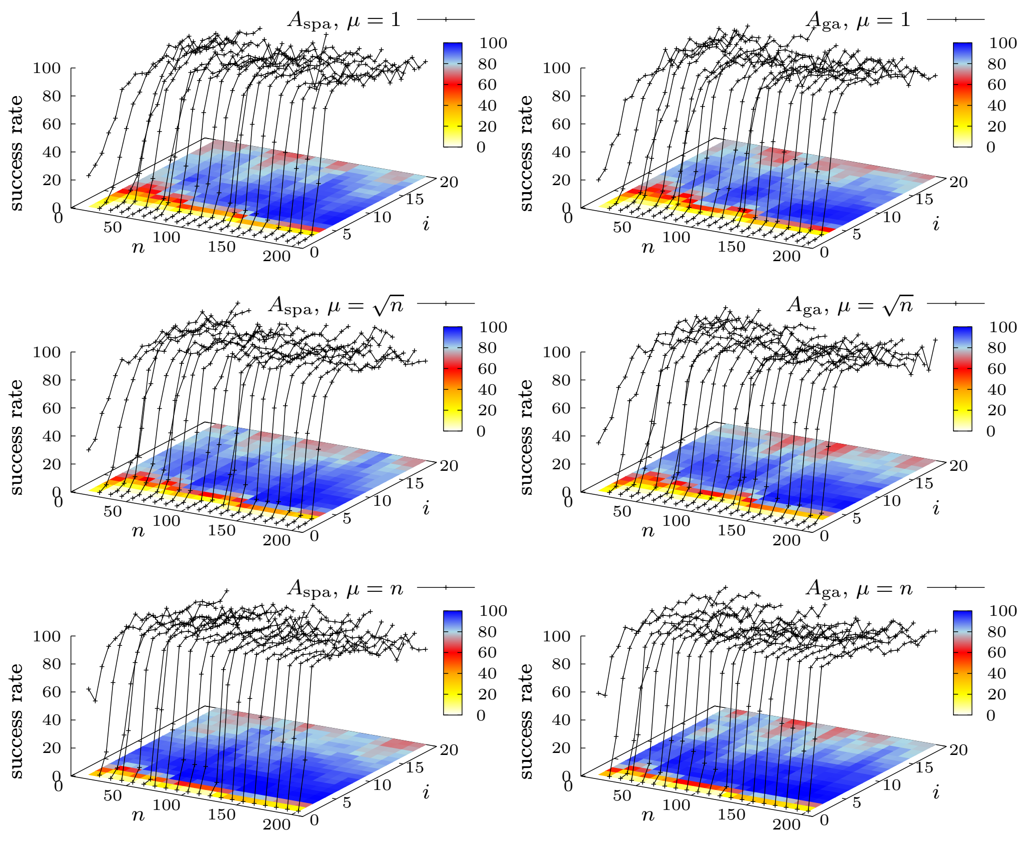


Figure 9.5.: Success rates in 100 independent runs of A_{spa} and A_{ga} on LocalOpt$_k$ with $k = 2$, different sizes μ of the collection of search points and decreasing values of $\tau_{\max}$.

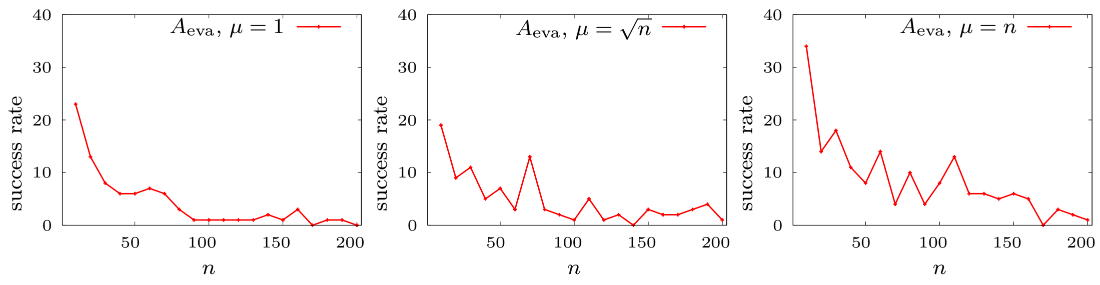


Figure 9.6.: Success rates in 100 independent runs of A_{eva} on $\textsc{LocalOpt}_k$ with $k = 2$, $\tau_{\max} = \lfloor 6 \log^2(n) \cdot (n^k + \mu n) \rfloor$, and different sizes μ of the collection of search points.

For the third set of experiments we now consider evolutionary aging. Due to Theorem 9.3 the optimization time is at least t with probability $1 - O\left(\left((\mu \log \mu)/n^{k-1}\right) + t/n^{(n/2)-k}\right)$ for some $t \in \mathbb{N}$. Moreover we have an expected optimization of $\Omega(2^n)$ if $\mu = o\left(n^{k-1}/\log n\right)$. We see that the condition on μ is not fulfilled for $\mu = n$ and that the first statement becomes trivial in that case since it only yields a probability of $\Omega(1)$. Thus, experiments are particularly interesting as they can give hints to the extensibility of the theorem.

Remember that we again fix the maximal number of iterations executed to the corresponding upper quartile from the first set of experiments. Thus, we can assume t to be polynomial and thus the theorem yields success probabilities of $O(1/n)$ and $O(\log(n)/\sqrt{n})$ respectively. The empirical results are shown in Figure 9.6. We see that in all three cases the success rate is decreasing with increasing n and is clearly smaller than 50%. Moreover, it decreases more slowly for bigger sizes of the collection of search points as indicated by Theorem 9.3. We assume that there is room for improvements concerning an extension to bigger collection of search points.

9.4.2. Results for Plateaus

For $\textsc{Plateau}$ we first consider evolutionary aging and genotypic aging. According to Theorem 9.7 an appropriate maximal lifespan when using evolutionary aging is $\tau_{\max} = \omega(\log(n) \cdot (n + \mu \log n))$. Analogously to the experiments for $\textsc{LocalOpt}_k$ we set $\tau_{\max} = \lfloor 6 \log^2(n) \cdot (n + \mu \log n) \rfloor$. In the case of genotypic aging Theorem 9.10 indicates $\tau_{\max} = \omega(\mu n \log n)$ and we choose $\tau_{\max} = \lfloor 6 \mu n \log^2 n \rfloor$. In both cases the optimization time is $O\left(\mu n^3\right)$. Thus, we get optimization times of $O\left(n^3\right)$ for $\mu = 1$, $O\left(n^{3.5}\right)$ for $\mu = \lfloor \sqrt{n} \rfloor$ and $O\left(n^4\right)$ for $\mu = n$.

The empirical results for this set of experiments are shown in Figure 9.7. Again we plot additionally $c \cdot b(n)$ if we have a bound of $O(b(n))$ on the expected optimization time for illustrative purposes where the constant c is determined using a least squares fit. We see that the optimization times in our experiments correspond pretty well to the proven asymptotic upper bounds. The variance within the 100 runs is not too big.

Moreover there is no significant difference between the two considered aging mechanisms evolutionary aging and genotypic aging.

As for LocalOpt$_k$ we examine the influence of the maximal lifespan on the optimization time. The results are shown in Figure 9.8. Here a fundamental difference between the two aging operators becomes obvious. On one hand evolutionary aging seems not to care much about the maximal lifespan since all success rate are around the expected value of 75. On the other hand genotypic aging with $\mu = 1$ and $\mu = \lfloor\sqrt{n}\rfloor$ shows the same behavior as seen in the experiments on LocalOpt$_k$ and we observe a drastic decrease of the success rate when τ_{max} becomes very small. For genotypic aging and $\mu = n$ this is again not the case. We conclude that evolutionary aging is much more robust against changes of the maximal lifespan whereas for genotypic aging this depends highly on the size μ of the collection of search points. The nature of this dependence on μ (described as a function depending on n) needs to be determined and is a subject of future research.

Finally we consider static pure aging on Plateau analogously to evolutionary aging on LocalOpt$_k$. As the maximal lifespan τ_{max} is different for evolutionary aging and genotypic aging we consider both values in our experiments. Due to Theorem 9.8 we know that the optimization time of A_{spa} is at least $n^{\sqrt{\alpha(n)}}$ with probability $1-n^{-\sqrt{\alpha(n)}}$ for some $\alpha(n) = \omega(1)$ and $\alpha(n) = O(n/\ln n)$. As $\tau_{\text{max}} = O\big(n^3/(\alpha(n)^{3/2}\ln n)\big)$ we can assume $\alpha(n) = \Omega(n/\ln n)$ for $\tau_{\text{max}} = \big\lfloor 6\log^2(n)\cdot(n+\mu\log n)\big\rfloor$ and $\tau_{\text{max}} = \big\lfloor 6\mu n\log^2 n\big\rfloor$ with $\mu = 1$. For $\tau_{\text{max}} = \big\lfloor 6\mu n\log^2 n\big\rfloor$ and $\mu = \lfloor\sqrt{n}\rfloor$ we can only assume $\alpha(n) = \Omega\big(n/\log^2 n\big)$. For $\tau_{\text{max}} = \big\lfloor 6\mu n\log^2 n\big\rfloor$ and $\mu = n$ only $\alpha(n) = \Omega\big(n^{2/3}/\log^2 n\big)$.

The results of the experiments are shown in Figure 9.9. It is obvious that the success rate drops rapidly to 0 for both values of τ_{max} since already for $n = 30$ it is constantly 0. In particular the success rate is clearly smaller than the success rate of evolutionary aging on LocalOpt$_k$ showing that static pure aging is really in trouble on Plateau.

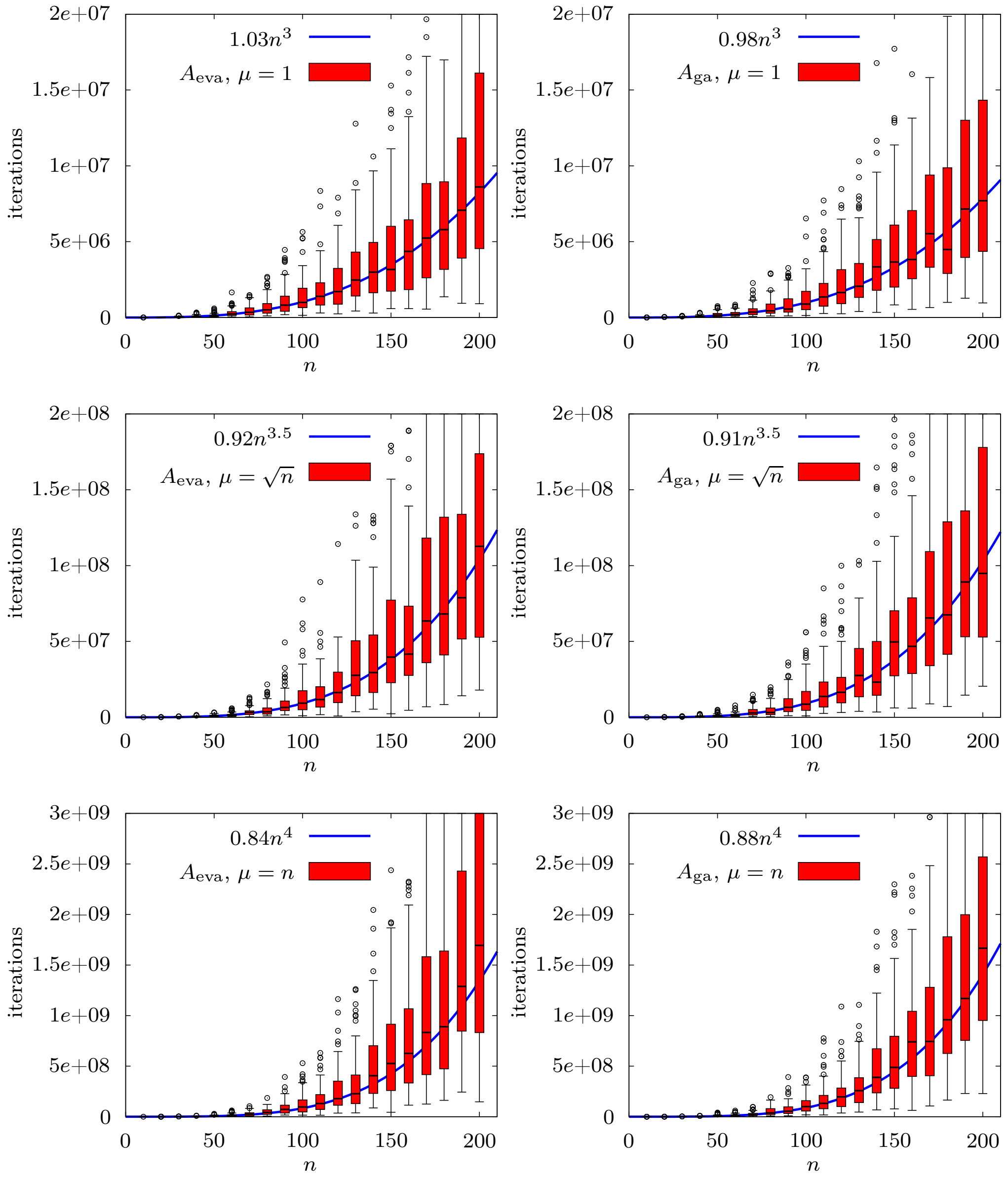


Figure 9.7.: Empirical results for A_{eva} and A_{ga} on PLATEAU with different sizes μ of the collection of search points, data from 100 runs.

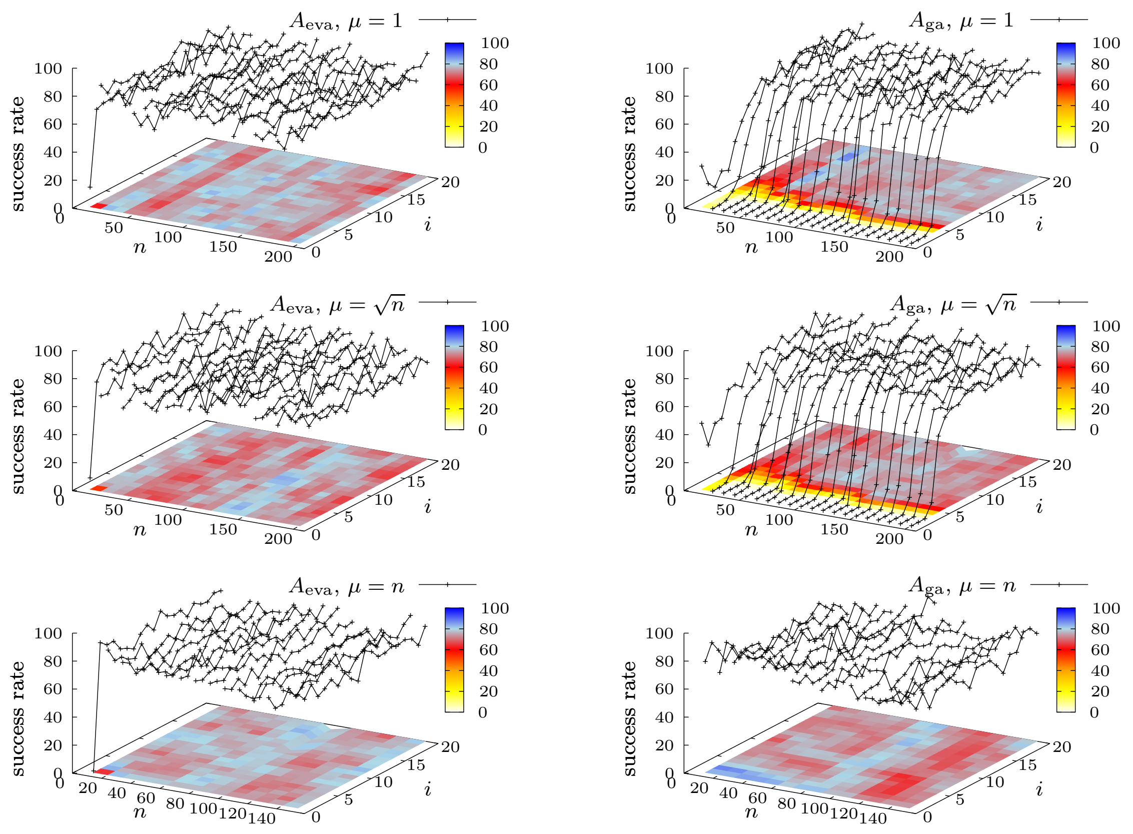


Figure 9.8.: Success rates in 100 independent runs of A_{eva} and A_{ga} on PLATEAU with different sizes μ of the collection of search points and decreasing values of $\tau_{\max}$.

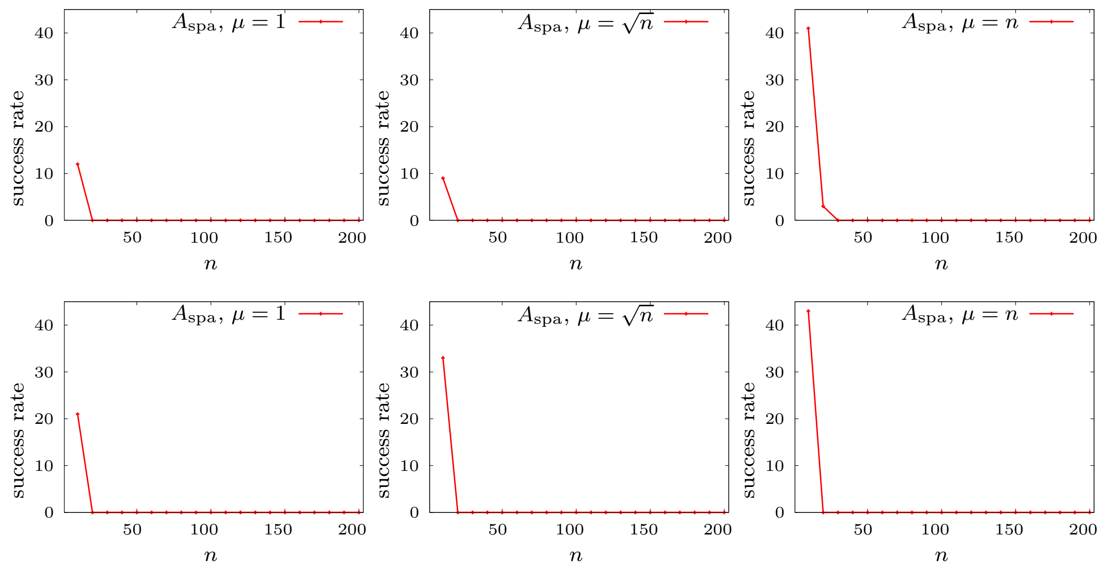


Figure 9.9.: Success rates in 100 independent runs of A_{spa} with $\tau_{\max} = \lfloor 6 \log^2(n) \cdot (n + \mu \log n) \rfloor$ from Theorem 9.7 (top) and $\tau_{\max} = \lfloor 6\mu n \log^2 n \rfloor$ from Theorem 9.10 (bottom) on PLATEAU with different sizes μ of the collection of search points.

10. Aging Beyond Restarts

In the previous chapter, we have seen that on one hand static pure aging can greatly increase the performance of a randomized search heuristic by recognizing stagnation and enabling it to perform restarts. Thus, it permits the algorithm to escape from local optima. On the other hand static pure aging does not achieve any benefits on plateaus but rather can be harmful in this situation since it mistakes missing progress in function values for stagnation. Even the result on the function containing a long k-path (Section 8.2) incorporates some kind of restart that is responsible for the success of the search heuristic.

While being able to perform restarts can be essential for the performance of the search heuristic it can also be achieved by simpler and computationally less expensive mechanisms. As seen in the previous chapter static pure aging performs a restart if all search points share the same age and all exceed the maximal lifespan $\tau_{\max}$. Since a new search point is assigned age 0 if it excels in the function value we can conclude that the last improvement occurred $\tau_{\max}$ rounds ago. In order to implement static pure aging each search point needs to be assigned an age which has to be adjusted in each round. Moreover, search points exceeding the maximal lifespan $\tau_{\max}$ need to be removed and replaced by new search points which makes additionally evaluations of the objective function necessary. We observe that it is computationally much cheaper to keep track of the number of rounds since the last improvement in function value occurred and to perform a complete restart if this number exceeds $\tau_{\max}$.

It is highly interesting to see what aging can achieve with respect to efficiency of an artificial immune system that cannot be achieved by restarts. One obvious difference is that aging may replace only parts of the current collection of search points by the newly generated search points. Such partial restarts may prove beneficial for certain problems. We address this question by first extending our algorithm framework in Section 10.1. We combine static pure aging with crossover, a variation operator known from evolutionary algorithms. Additionally, we introduce a mild strategy to maintain a certain level of diversity with respect to age. We consider different variants of static pure aging and selection for replacement mechanisms and investigate the interplay between this modules. In particular, we point out similarities and differences of the different algorithmic variants. One might wonder why we use static pure aging instead of genotypic aging. Since the problem considered in this chapter does not contain a plateau, both operators exhibit essentially the same behavior. We therefore stick to the operator currently used in practice.

The results in the remainder of this chapter are structured in results that hold for all considered variants of static pure aging (Section 10.2), results that depend on the aging strategy but are independent of the replacement strategy (Section 10.3), and results that depend on the concrete instantiation of static pure aging (Sections 10.4–10.7). We close the chapter by presenting an empirical evaluation of the algorithm for different sizes of the collection of search points (Section 10.8). The content of this chapter is based on the work done in Jansen and Zarges (2010a,b, 2011c).

10.1. A Structured View on Aging

We are interested in what aging can achieve beyond restarts and what influence age diversity mechanisms can have on the efficiency of the optimization. We analyze this by extending our algorithmic framework (Algorithm 7.1) with a more involved variation operator and different aging and replacement strategies. This way a more structured view on aging is established, allowing for an analysis of the interplay between aging and the selection mechanism incorporated into the algorithm. In the following, we first describe this extended framework. Afterwards, we introduce the objective function under consideration.

10.1.1. Further Extending the Framework

We consider Algorithm 7.1 and only present the concrete extensions here, i. e., the variation (line 2c), the decision on the new search point's age (line 2d), and the selection for replacement (line 2f). The other modules are defined just like in Section 7. Like before, we abbreviate Algorithm 7.1 using static pure aging by A_{spa}, independently of the concrete aging and replacement variant.

Variation

Variation (Algorithm 10.1) creates one new search point y by means of k-point crossover and standard bit mutations (Algorithm 3.5) known from evolutionary algorithms (de Jong 2006). The crossover operator is efficient when the collection of search points is sufficiently diverse. Since aging aims at increasing the diversity it is a good idea and interesting test case to combine crossover with aging. In k-point crossover two search points $x, y \in \{0,1\}^n$ are cut into $k+1$ pieces by selecting uniformly at random k *different* cut positions. A new search point is constructed from the pieces by taking all the odd numbered pieces of x (the first, third, ...) and all even numbered pieces of y (the second, fourth, ...) and concatenating them in increasing interleaving order. Usually, k-point crossover with very small values for k is employed, most often $k = 1$ or $k = 2$.

We apply these k-point crossover and standard bit mutations in the following way. With probability p_c (a parameter of the algorithm), we select two search points from C uniformly at random and perform k-point crossover. The result is subject to mutation. The final result is the new search point y. If no crossover is performed (with probability $1 - p_c$), we select one search point from C uniformly at random and mutate it, the result being the new search point y. The variation operator is formally defined in Algorithm 10.1.

Algorithm 10.1 Variation.

```
1.  **With probability** $p_c$
2.      Select $x_1, x_2 \in C_{t-1}$ uniformly at random.
3.      Select $c_1, c_2, \ldots, c_k \in \{0, 1, 2, \ldots, n\}$ pairwise different and uniformly at random.
4.      Sort $c_1, \ldots, c_k$ in ascending order and set $c_{k+1} := n + 1$.
5.      **If** $c_1 > 0$ **Then**
            Set $h := 1$ and $i := 0$.
        **Else**
            Set $h := 2$ and $i := 1$.
6.      **For all** $j \in \{0, \ldots, n-1\}$
7.          Set $y[j] := x_{i+1}[j]$.
8.          **If** $j \geq c_h$ **Then**
                Set $i := 1 - i$ and $h := h + 1$.
9.  **Else**
10.     Select $x_1 \in C_{t-1}$ uniformly at random.
11.     Set $y := x_1$ and $x_2 := x_1$.
12. Independently for each $i \in \{0, 1, \ldots, n-1\}$
13.     With probability $1/n$ set $y[i] := 1 - y[i]$.
```

Variants of Static Pure Aging

The basic idea of static pure aging is to assign age 0 if the new search point is an improvement. Otherwise it inherits its age from the search points it is derived from. As search points created by crossover have two search points as origin, things are less obvious in that case. It is unclear how the comparison with respect to the function value is to be made and what age is to be inherited if no improvement was made. One may believe that these are unimportant details as they only matter in the case of crossover and if the new search point is not good anyway. However, we will later see that this is not the case. We consider the general formulation of static pure aging from Algorithm 10.2 and implement three concrete variants described below.

Algorithm 10.2 Outline of Static Pure Aging.

```
1. **If** $f(y) > \max\{f(x_1), f(x_2)\}$ **Then**
2.     Set $y$.age := 0.
3. **Else**
4.     Set $y$.age := age of either $x_1$ or $x_2$.
```

The idea of static pure aging is to punish a new search point that fails to be an improvement by having it inherit its age. Improvements are rewarded by assigning age 0 and thus a longer lifespan. In the case of crossover the worst punishment possible is to assign the new search point y the larger age of the two other involved search points, x_1 and x_2.

While being simple it does not appear to be entirely fair. The reason the new search point fails to be an improvement could be that a good search point was combined with a bad search point. It therefore makes sense to compare the function values of x_1 and x_2. If these function values are equal we set the new search point's age to the older age. If, however, the two search points have different function values we have a choice. We can react in an optimistic way to this difference and assign the new search point the age of the better search point. Alternatively, we could be pessimistic and assign the new search point the age of the worse search point. We define all three variants formally in Definition 10.1.

Definition 10.1 (Jansen and Zarges (2011c)). *A new search point y that was either created by crossover of x_1 and x_2 or by mutation of x_1 (where we have $x_1 = x_2$ for notational simplicity) is assigned its age as outlined in Algorithm 10.2. Line 4 of this algorithm is detailed in three variants as follows.*

In age-based static pure aging *the age is set to the age of the older search point: $y.age := \max\{x_1.age, x_2.age\}$.*

In optimistic value-based static pure aging *the age is set to the age of the search point with larger function value, in case of equal function values to the larger age: If $f(x_1) \neq f(x_2)$ then $y.age :=$ argmax$\{f(x_1), f(x_2)\}.age$, else $y.age := \max\{x_1.age, x_2.age\}$.*

In pessimistic value-based static pure aging *the age is set to the age of the search point with smaller function value, in case of equal function values to the larger age: If $f(x_1) \neq f(x_2)$ then $y.age :=$ argmin$\{f(x_1), f(x_2)\}.age$, else $y.age := \max\{x_1.age, x_2.age\}$.*

Variants of Selection for Replacement

The selection for replacement is the part of the algorithm where age diversity mechanisms come into play. However, the function values are the more important selection criteria. If at least one current search point is removed due to its age the new search point is inserted. Otherwise it is only inserted if its function value is not worse than the worst function value of any of the current search points. If its function value is strictly larger than this value it replaces one current search point that is selected uniformly at random among all search points with worst function value. If its function value is equal to the worst function value, we have to be more careful. If age is considered to be helpful it makes sense to avoid having all current search points of the same age. We consider the general formulation for selection for replacement from Algorithm 10.3 and discuss four different concrete strategies described below.

We consider different variants of selection for replacement in order to investigate the role of age diversity for static pure aging. Probably the simplest way to maintain a certain degree of age diversity is to replace a search point whose age appears most frequently within the current collection of search points (including the new point itself). This mechanism ensures that the number of different age values among the worst search points does not decrease by exchanging two search points with worst function value. Note that it only affects the worst points in the current selection and only comes into play if another point with this worst function value is inserted. Another possible replacement strategy

Algorithm 10.3 Outline of Selection for Replacement.

```
1.  If $|C_t| < \mu$ then
2.      If $y$.age $\leq \tau_{\max}$ then
3.          Set $C_t := C_t \cup \{y\}$.
4.      While $|C_t| < \mu$ do
5.          Select $x \in \{0,1\}^n$ uniformly at random.
6.          Set $x$.age := 0. Set $C_t := C_t \cup \{x\}$.
7.  Else
8.      Choose $z \in C_t$ with minimal fitness uniformly at random.
9.      If $f(y) = f(z)$ then
10.         Determine a candidate set $D$ considering an appropriate replacement strategy.
11.         Select $x \in D$ uniformly at random.
12.         Set $C_t := (C_t \cup \{y\}) \setminus \{x\}$.
13.     If $f(y) > f(z)$ then
14.         Set $C_t := (C_t \cup \{y\}) \setminus \{z\}$.
```

removes a search point that is selected uniformly at random among all search points with minimal difference in age to the new search point. Again, it is ensured that the number of different age values among the worst search points does not decrease by exchanging two search points with worst function value.

We additionally consider two variants that do not employ age diversity mechanisms. On one hand, we analyze an algorithm that ignores the current age structure and simply replaces one current search point that is selected uniformly at random among all search points with worst function value. Note that this variant corresponds to the standard selection for replacement method in evolutionary algorithms. On the other hand, we consider the extreme case where age diversity is intentionally destroyed by replacing a search point whose age appears fewest within the current collection of search points (including the new point itself). Similar to the different static pure aging strategies we define a set of replacement strategies formally.

Definition 10.2 (Jansen and Zarges (2011c))**.** *A new search point y that was created during the variation phase of the algorithm replaces a search point from the current collection of search points C_t as outlined in Algorithm 10.3. Line 10 of this algorithm is detailed in four variants as follows. In all variants only search points with worst function value are considered for replacement, namely $D' := \{x \in C_t \mid f(x) = \min_{x' \in C_t} f(x')\}$.*

In most frequent replacement *the set of search points whose age occurs most frequently within the current selection of search points (including y) is determined. Formally, let*

$$f_a = |\{x \in (C_t \cup y) \mid x.age = a\}|$$

be the number of occurrences of age a and

$$f_{\max} = \max_{a \in \{0,1,\ldots,\tau_{\max}\}} f_a$$

the number of occurrences of the age that occurs most frequently in the current selection of search points. Note, that there may be multiple ages that occur most frequently in C_t. In this case, all these ages are taken into account. Then, $D = \{x \in C_t \mid f_{x.age} = f_{\max}\}$.

In smallest age distance replacement *a search point from D' with minimal age distance to the new search point is selected uniformly at random, i. e.,*

$$D = \left\{ x \in D' \mid |x.age - y.age| = \min_{x' \in D'} \left(|x'.age - y.age|\right) \right\}.$$

In random replacement *simply a search point from D' is selected for replacement uniformly at random, i. e., $D = D'$.*

In fewest replacement *the set of search points whose age occurs fewest within the current selection of search points (including y) is determined. Formally, let*

$$f_{\min} = \min_{a \in \{0,1,\dots,\tau_{\max}\}} f_a$$

be the number of occurrences of the age that occurs fewest in the current collection of search points. Again, there may be multiple ages that occur fewest in C_t. Then, $D = \{x \in C_t \mid f_{x.age} = f_{\min}\}$.

10.1.2. The Example Function Under Consideration

Since we want to see beneficial effects due to aging we again consider a function with a local optimum that is much easier to find than any global optimum. It is important that a global optimum cannot be found efficiently by means of an appropriate restart mechanism. The considered function $f\colon \{0,1\}^n \to \mathbb{R}$, formally defined in Definition 10.3, achieves all this and other goals. A visualization of f is given in Figure 10.1. Since

Definition 10.3 (Jansen and Zarges (2011c))**.** *For $n = 4k$, $k \in \mathbb{N}$, and $x \in \{0,1\}^n$ the function $f\colon \{0,1\}^n \to \mathbb{R}$ is defined by*

$$f(x) = \begin{cases} 2n & \text{if } x = 1^{n/4}0^{n/4}q,\ q \in \{0,1\}^{n/2}, |q|_1 \geq n/12, \\ n + i & \text{if } x = 1^i 0^{n-i},\ i \leq n/4, \\ n - \textsc{OneMax}(x) & \text{otherwise.} \end{cases}.$$

For the vast majority of the points x in the search space the function value is defined as $n - \textsc{OneMax}(x)$ leading the algorithm in direction of the all-zero bit string 0^n. This is the beginning of a path of Hamming neighbors of the form 1^i0^{n-i} (compare the function Ridge from Definition 8.9). As seen before, with such construction the local optimum $1^{n/4}0^{3n/4}$ is easy to find.

Points of the form $1^{n/4}0^{n/4}q$ with $q \in \{0,1\}^{n/2}$ and $|q|_1 \geq n/12$ are special. The set of all these points

$$\text{OPT} := \left\{1^{n/4}0^{n/4}q \mid q \in \{0,1\}^{n/2}, |q|_1 \geq n/12\right\}$$

equals the set of all global optima of f. The crucial observation is that these points are easy to locate by means of a k-point crossover of the local optimum $1^{n/4}0^{3n/4}$ and some

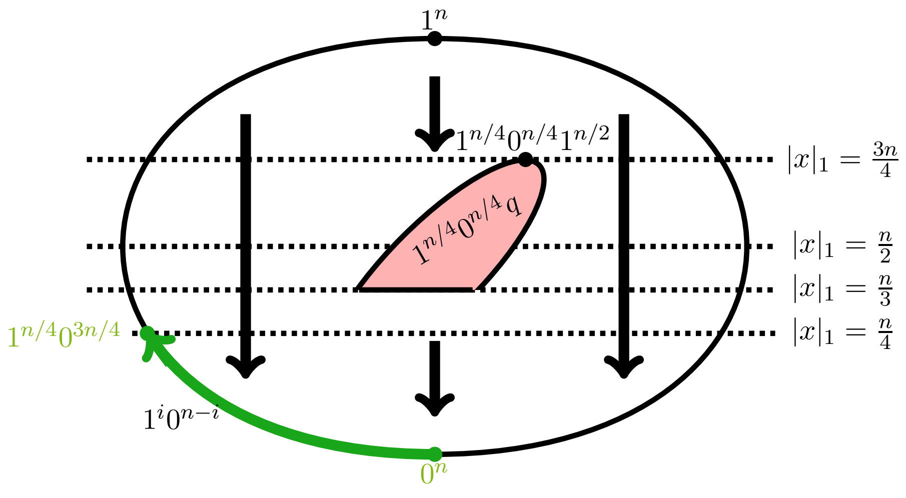


Figure 10.1.: The example function f. (Jansen and Zarges 2011c).

$y \in \{0,1\}^n$ that is chosen uniformly at random but very difficult otherwise. We make this precise in the following lemma.

Lemma 10.4 (Jansen and Zarges (2011c))**.** *Let $x = 1^{n/4}0^{3n/4}$ and $y \in \{0,1\}^n$ be selected uniformly at random. Let OPT be the set of global optima of f. Then, for any $k = O(1)$*

$$\mathit{Prob}\,(k\text{-}\mathit{Point}\text{-}\mathit{Crossover}(x,y) \in \mathit{OPT}) = \Omega(1)$$

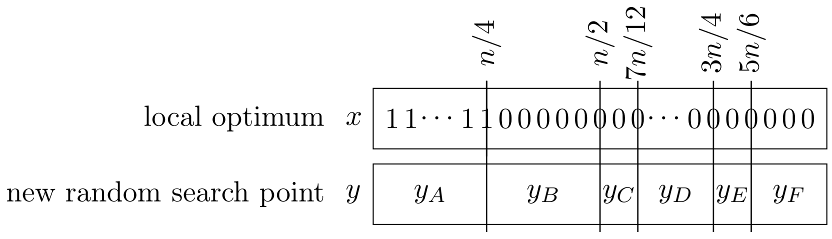
holds.

Proof. Optimal search points are of the form $1^{n/4}0^{n/4}q$ where $q \in \{0,1\}^{n/2}$ and $|q|_1 \geq n/12$. We consider one possible way of constructing $z \in \text{OPT}$ by means of k-point crossover of x and y and prove that this happens with a probability that is bounded below by a positive constant. Note that we do not aim at deriving tight bounds on this probability and sacrifice pointless accuracy in favor of simplicity of the proof.

Consider a crossover point c with $c \in \{0,1,\ldots,n\}$ selected uniformly at random. For any constants $0 \leq \delta < \delta' \leq 1$ we have that $\delta n \leq c < \delta' n$ holds with probability at least $\varepsilon > 0$ where ε is a positive constant depending on $\delta' - \delta$.

Consider crossover points $c_1 < c_2 < \cdots < c_k \in \{0,\ 1,\ \ldots,\ n\}$ (with $k \in \mathbb{N}$ and $k = O(1)$) selected uniformly at random. See Figure 10.2 for an illustration. We have $(1/2)n \leq c_1 < (7/12)n$ ($c_1 \in y_C$), $(3/4)n \leq c_2 < (5/6)n$ ($c_2 \in y_E$), and $(5/6)n \leq c_i \leq n$ ($c_i \in y_F$) simultaneously for all $i \in \{3,4,\ldots,k\}$ with probability at least ε^k where $\varepsilon > 0$ is some constant. Note that this holds for any constant k (and that for very small values of k like $k = 1$ the conditions on c_2 and c_i with $i > 2$ are empty and thus trivially hold).

In this situation the crossover of $x = 1^{n/4}0^{n/4}$ and y (where $y \in \{0,1\}^n$ uniformly at random) is carried out as can be seen in Figure 10.2. Having $(1/2)n \leq c_1 < (7/12)n$ implies that the leftmost $n/2$ bits of z equal $1^{n/4}0^{n/4}$ since these bits are copied from x.



$n/4$
$n/2$
$7n/12$
$3n/4$
$5n/6$
local optimum x
$11\cdots 1\,1\,000000\,0\,0\,0\cdots 0\,0\,0\,0\,0000$
new random search point y
y_A
y_B
y_C
y_D
y_E
y_F

Figure 10.2.: Visualization of x and y from Lemma 10.4 (Jansen and Zarges 2011c).

The bits between c_1 and c_2 ($c_2 = n$ in the case of 1-point crossover) are copied from y. Since we have $(1/2)n \leq c_1 < (7/12)n$ and $(3/4)n \leq c_2 < (5/6)n$ we know that these are at least $(3/4)n - (7/12)n = (1/6)n$ bits copied from y. Clearly, these bits are distributed uniformly at random. The expected number of 1-bits in these $(1/6)n$ bits equals $(1/12)n$ and we have at least $(1/12)n$ 1-bits among these bits with probability at least 1/2. Thus, we have $\text{Prob}\,(z \in \text{OPT}) \geq (1/2) \cdot \varepsilon^k = \Omega(1)$ as claimed. □

It is easy to see that the global optima of f are difficult to find in a different way. For all points $x \in \text{OPT}$ we have $n/3 \leq |x|_1 \leq (3/4)n$ and thus there are always exponentially many points with the same number of 1-bits. For each number i of 1-bits let OPT_i denote the set of bit strings from OPT with this number of 1-bits, i. e., $\text{OPT}_i = \{x \in \text{OPT} \mid |x|_1 = i\}$. Let $\overline{\text{OPT}_i} = \{x \in \{0,1\}^n \mid |x|_1 = i\} \setminus \text{OPT}_i$ denote the other strings with the same number of 1-bits. Clearly, we have $|\text{OPT}_i| \,/\, \left|\overline{\text{OPT}_i}\right| = 2^{-\Omega(n)}$ and we conclude that it is highly unlikely to find OPT by pure random sampling. This implies that restarts do not help. Also randomized search heuristics that are efficient on OneMax are unlikely to encounter OPT since they quickly leave the part of the search space with these numbers of 1-bits. Thus, they sample only a polynomial number of such bit strings and encounter OPT only with probability $n^{O(1)} \cdot 2^{-\Omega(n)} = 2^{-\Omega(n)}$.

Note that we do not claim that no search heuristic without aging can be efficient on f. Clearly, search heuristics choosing some $x \in \text{OPT}$ as initial search point optimize f with a single function evaluation. While such search heuristics obviously cheat on f by incorporating too much knowledge about it into their 'search strategy' there are other mechanisms that 'cheat' in less obvious ways. Mechanisms that maintain a high degree of diversity in the collection of search points are another way of coping with multi-modal problems. Friedrich et al. (2009b) consider several such mechanisms, among those one that preserves diversity on the level of fitness values. This mechanism works well on many functions where the number of function values is small, i. e., polynomially bounded. Clearly, for noisy functions or continuous functions in $\mathbb{R}^n$ such a mechanism cannot work. Moreover, for Ackley's well-known trap function (Ackley 1987) it achieves highly efficient optimization, a clear indication that this mechanism is cheating in some way. When discussing example functions it is completely pointless to discover or, even worse, invent search heuristics that are efficient on the example function under consideration. The point of considering example functions is to exhibit situations that highlight the usual working patterns of commonly used randomized search heuristics. While randomized search heuristic are very often used in practice and aging is a commonly used mechanism to improve their performance on difficult problems this fitness-based diversity mechanism is not commonly used. We therefore claim that reasonable search heuristics without aging and crossover fail to be efficient on this example function f.

The example function f as defined in Definition 10.3 is very specific. It is used as a vehicle to demonstrate and formally prove a number of properties of the aging operators we consider here. The same effects can be observed when optimizing other problems, too. For example, it is not essential that f contains exactly one local optimum where a crossover with random search points is needed to locate the global optimum. Functions with several of such local optima would not be very much different. However, it is essential that a partial restart is needed for the optimization. If a complete restarts suffices aging is not needed and may be replaced by an appropriate restart strategy.

We have seen that appropriate k-point crossover operations can yield a global optimum of f with good probability. In the following we will investigate the probability that the algorithms under consideration perform such crossover operations. If it is sufficiently large, the respective algorithm is efficient on f.

10.2. Common Properties of All Considered Aging Variants

As seen in Chapter 8, the most critical parameter of aging is the maximal lifespan $\tau_{\max}$. The results in that chapter indicate that a lower bound of $\tau_{\max} = \omega(\mu n \log \mu)$ suffices for all upper bounds on the expected optimization time here. Apart from this the algorithm as well as our proofs work for most settings of the other parameters. We require $\mu \geq 2$ since we need crossover and the usual bound $\mu = n^{O(1)}$, thus the size of the collection of search points μ is almost completely unrestricted. The crossover probability p_c can be set almost arbitrarily. It is usually set to some rather large constant like $p_c = 0.8$. Sometimes, very small crossover probabilities are used in proofs (see for example Jansen and Wegener (2002)) but in practice this is hardly ever done. We deal with any value $0 < \varepsilon \leq p_c \leq 1-\varepsilon < 1$ for some arbitrarily small constant $\varepsilon > 0$. Setting p_c in this way the concrete value of p_c has no influence on the asymptotic expected optimization time. Note, however, that having p_c converge to either 0 or 1 may change things considerably. While it is not difficult to adjust our upper bounds to such settings we refrain from considering these rather unusual cases. As already pointed out in Lemma 10.4 the number k of crossover points used in k-point crossover is not very important as long as it is bounded above by a constant. Clearly, smaller values are better for f and we restrict our attention to the commonly used 1-point crossover in Section 10.8 when performing experiments.

The different variants of A_{spa} behave very similarly until the local optimum is reached for the first time. The main difference of the considered variants is the way a global optimum can be constructed by recombination of a local optimum and a randomly chosen search point like in Lemma 10.4. This can happen when we have at least two search points in the local optimum and some but not all of those locally optimal search points are removed due to their age. We call such an event a *partial restart*. It is unclear how likely it is that such a partial restart occurs. Moreover, we need to derive the probability that given that a partial restart occurs an appropriate crossover operation is executed afterwards. We leave these two questions open for the moment and first prove parameterized lower and upper bounds for all strategies. In these bounds the probabilities for these two events appear as unknowns. Afterwards, we investigate them separately for the different replacement strategies. We start with the upper bound on the optimization time of A_{spa} and the parameter settings discussed above.

Lemma 10.5 (Jansen and Zarges (2011c))**.** *Let $\mu \in \mathbb{N}$ with $\mu \geq 2$ and $\mu = n^{O(1)}$, p_c with $0 < \varepsilon \leq p_c \leq 1 - \varepsilon < 1$ for a positive constant ε, $k = O(1)$, and $\tau_{\max} = \omega(\mu n \log \mu)$. Moreover, let p denote the probability that a partial restart occurs and q the probability for an appropriate crossover operation creating the global optimum after such partial restart.*

The expected optimization time of A_{spa} using an arbitrary strategy for static pure aging from Definition 10.1 and an arbitrary strategy for selection for replacement from Definition 10.2 on f (Definition 10.3) is

$$E\big(T_{A_{spa},f}\big) = O\big(p^{-1}q^{-1}\left(\tau_{\max} + n^2 + \mu n \log n\right)\big).$$

The probabilities p and q depend on n, μ, p_c, k and $\tau_{\max}$ and the aging and replacement strategies used. For the sake of readability we just write p and q since the parameters are obvious from the context.

Proof of Lemma 10.5. There are three regions of the search space that correspond to phases of a run of A_{spa} on f. In the vast majority of the search space the fitness value is given as $n - \textsc{OneMax}(x)$. As seen in previous proofs, this part can be optimized as the (μ+1) EA on OneMax due to our lower bound on the maximal lifespan $\tau_{\max}$. The additional use of crossover cannot increase the asymptotic growth of the expected optimization time here since with probability $1-p_c \geq \varepsilon = \Omega(1)$ no crossover is performed. Thus the expected optimization of $O(\mu n + n \log n)$ (Witt 2006) carries over. Second, there are the bit strings of the form $1^i 0^{n-i}$ with $i \leq n/4$. Again, due to our lower bound on the maximal lifespan $\tau_{\max}$ this part can be optimized as the (μ+1) EA optimizes LeadingOnes, i. e., in expected time $O\big(n^2 + \mu n \log n\big)$ (Witt 2006).

Finally, we are interested in constructing a global optimum by recombination of a local optimum and a randomly chosen search point like in Lemma 10.4. This happens with probability q after a partial restart that in turn occurs with probability p. Both experiments follow the geometric distribution (Lemma B.14). Thus, after expected q^{-1} partial restarts a globally optimal search point is created. Moreover, A_{spa} requires expected p^{-1} trials to perform a partial restart.

Clearly, the time we need to wait until a search point is removed due to its age (or earlier due to other reasons) is at most $\tau_{\max}$ which concludes the proof. □

The following lower bound on the optimization time of A_{spa} holds for all possible values of $\tau_{\max}$ and the same settings of μ and p_c as before.

Lemma 10.6 (Jansen and Zarges (2011c))**.** *Let $\mu \in \mathbb{N}$ with $\mu \geq 2$ and $\mu = n^{O(1)}$, p_c with $0 < \varepsilon \leq p_c \leq 1 - \varepsilon < 1$ for a positive constant ε, $k = O(1)$, and $\tau_{\max} = 2^{O(n)}$.*

Moreover, let p denote the probability that a partial restart occurs and q the probability for an appropriate crossover operation creating the global optimum after such partial restart.

The expected optimization time of A_{spa} using an arbitrary strategy for static pure aging from Definition 10.1 and an arbitrary strategy for selection for replacement from Definition 10.2 on f (Definition 10.3) is

$$E\left(T_{A_{spa},f}\right) = \Omega\left(p^{-1}q^{-1}\left(\tau_{\max} + n^2 + \mu n \log n\right)\right).$$

Proof. As in the proof of Lemma 10.5 there are three regions in the search space that correspond to phases of a run of A_{spa} on f: the part where the fitness value is given by $n - \text{OneMax}(x)$, the LeadingOnes-like path to the local optimum and the region of the global optimum.

First assume that the maximal lifespan $\tau_{\max}$ is sufficiently large, say $\omega(\mu n \log \mu)$. In this case a lower bound can be proven similarly to the (μ+1) EA on LeadingOnes (Witt 2006). For this we need to show that the LeadingOnes-like path is first reached by a search point with a number of 0-bits that is $\Omega(n)$. We pick $(7/8)n$ here somewhat arbitrarily.

As already discussed in Section 10.1.2 the probability that the algorithm initializes in some $x \in \text{OPT}$ is $2^{-\Omega(n)}$. Moreover, the probability to encounter OPT by optimizing the $n - \text{OneMax}(x)$ part in the first phase is $2^{-\Omega(n)}$. Analogously we can show that A_{spa} first hits the path to the local optimum with a search point with at most $n/8$ 1-bits with probability $1 - 2^{-\Omega(n)}$. Let L_i denote the set of bit strings with i 1-bits. Then, the local optimum belongs to $L_{n/4}$ and the probability that A_{spa} reaches the path at some point with at least $n/8$ 1-bits is

$$(n/8)/\sum_{i=1}^{n/8} |L_i| = 2^{-\Omega(n)}.$$

Note that crossover does not increase the probability of finding the path with a larger number of 1-bits as after initialization all search points have at least $n/16$ 0-bits within the first $n/4$ of the bit string with probability $2^{-\Omega(n)}$ and the number of 0-bits is increasing during the $n - \text{OneMax}(x)$ phase.

We now investigate the probability to make some progress on the path. Assume, there are b best search points on the path. Then, the probability to create a new best search point on the path by means of mutation is $O(b/(\mu n))$ as one of the b best search points has to be selected and at least a mutation of a single bit is needed.

Considering a crossover operation of a search point $x = 1^i 0^{n-i}$ on the path and a search point y that has not yet reached the path, we easily get the same upper bound. In order to create another point on the path, x has to be selected as first parent which happens with probability at most b/μ. Moreover, $y_{i+1} = 1$ is needed to increase the number of leading 1-bits and thus, yield progress on the path. This event has probability at most $1/2$. Finally, we require $c_1 = i$ for the first crossover point in order to copy the old i leading 1-bits from x and the additional one from y. This happens with probability at most $1/n$.

Clearly, it is not possible to create a new best search point with crossover of two search points on the path. Points on the path have the form 1^i0^{n-i} and if the crossover of 1^i0^{n-i} and 1^j0^{n-j} yields another path point then this path point is 1^k0^{n-k} with $\min\{i,j\} \leq k \leq \max\{i,j\}$. In particular, this holds for the local optimum. Remember that at the point of time where the first search point in the local optimum is created all other members of the collection of search points are on the path with probability close to 1. Thus, the first search point in the local optimum has to be created by means of mutation and thus, gets age 0.

We still need to consider if crossover can asymptotically decrease the time we need to increase the number of best search points on the path from 1 to b. The probability to increase this number from b to $b+1$ by means of mutation is $\Theta(b/\mu)$ since we need to select one of the b best search points and do not flip any bits during mutation which happens with probability $1/4 \leq (1-1/n)^n \leq 1/e$. For crossover it is necessary to select a currently best search point as first parent and thus, the probability to create a copy by means of crossover is also $O(b/\mu)$.

Altogether, we see that the lower bound for the LeadingOnes part carries over from the $(\mu{+}1)$ EA (Witt 2006) and we get $\Omega\big(n^2 + \mu n \log n\big)$ for the second phase. Note, that this dominates the upper bound for the first phase.

Once the complete collection of search points is on the path to the local optimum or in the local optimum, i. e., of the form 1^i0^{n-i} for possibly different values of i with $1 \leq i \leq n/4$ for all of them, the global optimum can only be reached via a direct mutation to the OPT region. Such a mutation has probability at most

$$\binom{n}{n/12} \cdot \left(\frac{1}{n}\right)^{n/12} \leq \frac{1}{(n/12)!}$$

as at least $n/12$ bits in the second half of the bit string have to flip. Thus, the probability to create a global optimum by means of mutation is $n^{-\Omega(n)}$. Clearly, it is not possible to create a global optimum with crossover of two search points on the path.

As the waiting time for a partial or complete restart is at least $\tau_{\max}$, we need time $\Omega\big(\tau_{\max} + n^2 + \mu n \log n\big)$ to get into a situation where the first search point in the local optimum is removed due to the maximal lifespan. As in Lemma 10.5 we need expected q^{-1} partial restarts to create a globally optimal search point and expected p^{-1} trials to perform a partial restarts which proves the claimed lower bound in the case where the maximal lifespan $\tau_{\max}$ is sufficiently large.

If the maximal lifespan $\tau_{\max}$ is not sufficiently large the search process is slowed down as it becomes harder to reach the local optimum. If $\tau_{\max}$ is very small almost constantly new search points are created uniformly at random. In t time steps, at most $t\mu/\tau_{\max}$ new search points are created in this way. Each of these new search points is equal to a specific globally optimal search point with probability 2^{-n} as discussed above. Such a process finds some of the less than $2^{n/2}$ global optima in an expected number of more than $2^{n/2}$ steps. □

10.3. Properties of Static Pure Aging Independently of the Replacement Strategies

Before considering the different replacement strategies, we further discuss the concept of partial restarts, i. e., the event when we have at least two search points in the local optimum and some but not all of those locally optimal search points are removed due to their age. A necessary condition for such an event is that by the time when the maximal lifespan $\tau_{\max}$ is reached by one of the search points in the local optimum, at least two search points with different ages are in the local optimum. Moreover, there is no possibility to create another locally optimal search point with different age once all search points have reached the local optimum as descendants always inherit the age of one of their parents. Thus, the two search points in the local optimum with different ages have to be created while the collection of search points approaches the local optimum. Additionally, after all search points have reached the local optimum, this property of the age structure within the collection of search points has to be preserved by the replacement strategy until the maximal lifespan $\tau_{\max}$ is reached by one of the search points in the local optimum.

The first condition, i. e., creating two search points with two different ages in the local optimum, is independent of the replacement strategy. We therefore analyze the probability for this event and the three static pure aging variants before looking closer at the different replacement strategies.

Lemma 10.7 (Jansen and Zarges (2011c)). *Let $\mu \in \mathbb{N}$ with $\mu \geq 2$ and $\mu = n^{O(1)}$, p_c with $0 < \varepsilon \leq p_c \leq 1 - \varepsilon < 1$ for a positive constant ε, $k = O(1)$, and $\tau_{\max} = \omega(\mu n \log \mu)$. Moreover, let p_1 be the probability for the event that two search points with different ages enter the local optimum before the whole collection of search points has reached the local optimum.*

For A_{spa} using age-based or pessimistic value-based static pure aging from Definition 10.1 and an arbitrary strategy for selection for replacement from Definition 10.2 on f (Definition 10.3), we have

$$p_1 = \Omega(1).$$

Proof. Consider a point of time when $\max\{f(x) \mid x \in C\}$ is increased to $(5/4)n$, i. e., a first locally optimal search point x is produced. Clearly, this search point enters the collection of search points and is assigned age 0. Note, that at this point of time all other search points have age different from x. At later points of time descendants of x may have the same age.

We claim that at this point of time all other members of the collection of search points also have form $1^i 0^{n-i}$ for different values of i with $i < n/4$ with probability close to 1. In the proof of Lemma 10.6 we saw that with probability close to 1 the first search point to enter the global optimum does so from the path and needed $\Omega\big(n^2 + \mu n \log n\big)$ steps to get there. Since all points on the path have larger function value than the other search points the probability to increase the number of search points on the path from some value v to $v+1$ is $\Omega(v/\mu)$: it suffices to select one such search point (probability v/μ) and do not change it (probability $(1-p_c)(1-1/n)^n = \Omega(1)$). Thus, the process of getting the whole collection of search points on the path has expected length $O(\mu \log \mu)$ (similar to the coupon collector process (Lemma B.17)), much smaller than the $\Omega\big(n^2 + \mu n \log n\big)$ steps needed in expectation to reach the local optimum. This implies the claim.

We consider the following $\tau_{\max} = \omega(\mu n \log \mu)$ iterations. We prove that within these $\tau_{\max}$ steps with probability $p = \Omega(1)$ another locally optimal search point with age different from x.age enters the collection of search points. To this end, we consider crossover.

Consider $x = 1^{n/4}0^{3n/4}$ and $y = 1^i 0^{n-i}$ with $0 \le i < n/4$. We have

$$\text{Prob}\,(k\text{-Point-Crossover}(x, y) = x) = \Omega(1)$$

in the same way as we obtained Lemma 10.4. Note that the offspring has the same fitness as x. Thus, in the pessimistic value-based variant, the offspring's age is set to the age of y which is different to the age of x, proving $p_1 = \Omega(1)$ in that case.

For the age-based variant we need to be slightly more careful since the offspring gets initial age $\max\{x.\text{age}, y.\text{age}\}$ and hence, it is not clear whether x is older than y or vice versa. When x entered the collection of search points it was the search point with minimal age 0. Thus, all other search points have age different from x unless they are created as descendants of x. We now consider the following $\Theta(\mu)$ steps. Clearly, in these steps this first search point x or one of its copies is selected for reproduction involving crossover with probability $\Omega(1)$. The expected number of descendants of x made in these $\Theta(\mu)$ steps is bounded above by $O(1)$ with probability $1-\delta$ for any constant $\delta > 0$. The number of search points in the collection of search points that may have improved within these steps is bounded by $O(\mu/n)$ since improvements can only occur via mutations but not by crossover alone as seen in the proof of Lemma 10.6. Thus, we only have $O(\mu/n)$ improved search points within these steps and this also holds with probability $1-\delta$ for any constant $\delta > 0$. We conclude that there are $\Omega(\mu)$ search points on the path with age larger than x. Thus, one of these is selected together with x with probability $\Omega(1)$ in this $\Theta(\mu)$ steps we consider. These two parents produce another locally optimal offspring that will have age different from x with probability $\Omega(1)$. □

For the considered problem, the only difference between aging in the optimistic and pessimistic value-based variant is the way partial restarts can be achieved, i. e., the way a second age can enter the local optimum. The main difference is that crossover no longer helps in creating another locally optimal search point with age different from the first search point entering the local optimum. If we perform crossover of $x = 1^{n/4}0^{3n/4}$ and $y = 1^i 0^{n-i}$ with $i < n/4$ the age of the new search point is given by the age of the better search point, i. e., by x.age. This is no different from a copy of x. Thus, we need to rely on mutations only.

Lemma 10.8 (Jansen and Zarges (2011c)). *Let $\mu \in \mathbb{N}$ with $\mu \ge 2$ and $\mu = n^{O(1)}$, p_c with $0 < \varepsilon \le p_c \le 1-\varepsilon < 1$ for a positive constant ε, $k = O(1)$, and $\tau_{\max} = \omega(\mu n \log \mu)$. Moreover, let p_1 be the probability for the event that two search points with different ages* *enter the local optimum before the whole collection of search points has reached the local optimum.*

For A_{spa} using optimistic value-based static pure aging from Definition 10.1 and an arbitrary strategy for selection for replacement from Definition 10.2 on f (Definition 10.3), we have

$$p_1 = \begin{cases} \Theta\left(\frac{\mu \log \mu}{n}\right) & \text{if } \mu \leq \delta \cdot \frac{n}{\log n} \text{ for constant } \delta > 0 \text{ sufficiently small} \\ \Theta(1) & \text{otherwise} \end{cases}.$$

Proof. Consider the first search point x that enters the local optimum.

We show that by the time half of the collection of search points is taken over by copies of x the rest of the collection of search points are all of the form $1^{(n/4)-1}0^{3(n/4)+1}$ with probability close to 1. If there are b copies of x the probability to increase the number of copies to $b+1$ is $O(b/\mu)$. Since initially we have $b = 1$ we obtain $\Omega(\mu \log \mu)$ as lower bound for creating $\mu/2$ copies of x. On average in these steps already copies of the second best have been produced. Since the selection for reproduction is uniform these copies are selected with higher probability than the first single best. This yields that all worse search points will be removed.

If there are b copies of a second best search point the probability to create a better search point is $O(b/(\mu n))$ since one of them has to be selected and at least a mutation of a single bit is needed. However, in the situation described above and as long as $b = \Theta(\mu)$, the probability to create another locally optimal search point via mutation is $\Omega(1/n)$. The expected time for increasing the number of copies from $b = \mu/2$ to $b = c\mu$ for some constant $c > 1/2$ is also $\Theta(\mu \log \mu)$. Again, this holds due to the similarity to the coupon collector process (Lemma B.17). Hence, the probability to create another locally optimal search point with age different from x is $\Omega(1/n)$ for $\Theta(\mu \log \mu)$ steps. After this number of steps this probability can decrease even to 0 since after that time the whole collection of search points may be in the local optimum. In the following we use c/n (for some sufficiently small positive constant c) as lower bound on this probability.

We start with the lower bound on the probability p_1. Assume $\mu \leq n/(cc' \log n)$, i. e., $\delta \leq 1/(cc')$, for positive constants c and c'. The probability that another locally optimal search point with age different from x is created within these $\Theta(\mu \log \mu)$ steps is

$$p_1 \geq 1 - \left(1 - \frac{c}{n}\right)^{c'\mu \log \mu} \overset{\text{(B.8)}}{\geq} 1 - e^{-cc'\frac{\mu \log \mu}{n}} \overset{\text{(B.9)}}{\geq} 1 - \frac{1}{1 + cc'\mu \log(\mu)/n}$$
$$= 1 - \frac{n}{n + cc'\mu \log \mu} = \frac{cc'\mu \log \mu}{n + cc'\mu \log \mu} \geq \frac{cc'\mu \log \mu}{2n}.$$

Otherwise we get

$$p_1 \geq 1 - \left(1 - \frac{c}{n}\right)^{c'\mu \log \mu} \overset{\text{(B.8)}}{\geq} 1 - e^{-cc'\frac{\mu \log \mu}{n}} = 1 - e^{-\Omega(1)} = \Omega(1).$$

Since p_1 is a probability (and thus $p_1 \leq 1$), $p_1 = \Omega(1)$ implies $p_1 = \Theta(1)$.

We still need to consider the upper bound for $\mu \leq \delta n/\log n$. The probability to create another locally optimal search points is at most $1/n$ as at least a mutation of a single bit is needed. Again, assume $\mu < n/\log n$. Then, $\mu \log \mu/(n-1) \leq 1$ holds. We see analogously to the calculations above that another locally optimal search point with different age is created in $\Theta(\mu \log \mu)$ steps with probability

$$\begin{aligned} p_1 \leq 1 - \left(1 - \frac{1}{n}\right)^{c\mu \log \mu} &\overset{(B.8)}{\leq} 1 - e^{-\frac{c\mu \log \mu}{n-1}} \\ &\overset{(B.9)}{\leq} \frac{2c\mu \log(\mu)/(n-1)}{1 + 2c\mu \log(\mu)/(n-1)} = \frac{2c\mu \log \mu}{n - 1 + 2c\mu \log \mu} \leq \frac{2c\mu \log \mu}{n} \end{aligned}$$

for some positive constant c, concluding the proof of the lemma. □

10.4. Smallest Age Distance Replacement Strategy

We are now ready to consider the different variants of selection for replacement from Definition 10.2. We start with the smallest age distance replacement strategy and prove upper and lower bounds on the expected optimization time using the results derived in the previous sections.

Theorem 10.9 (Jansen and Zarges (2011c)). *Let $\mu \in \mathbb{N}$ with $\mu \geq 2$ and $\mu = n^{O(1)}$, p_c with $0 < \varepsilon \leq p_c \leq 1 - \varepsilon < 1$ for a positive constant ε, $k = O(1)$, and $\tau_{\max} = \omega(\mu n \log \mu)$.*

The expected optimization time of A_{spa} using smallest age distance replacement from Definition 10.2 on f (Definition 10.3) is

$$E\left(T_{A_{spa},f}\right) = O\big(\mu \cdot \left(\tau_{\max} + n^2 + \mu n \log n\right)\big)$$

for age-based and pessimistic value-based static pure aging and

$$E\left(T_{A_{spa},f}\right) = O\left(\left(\mu + \frac{n}{\log \mu}\right) \cdot \left(\tau_{\max} + n^2 + \mu n \log n\right)\right)$$

for optimistic value-based static pure aging (Definition 10.1).

Proof. From Lemma 10.5 we know that $\mathrm{E}\left(T_{A_{\mathrm{spa}},f}\right) = O\big(p^{-1}q^{-1}\left(\tau_{\max} + n^2 + \mu n \log n\right)\big)$ holds where p is the probability that a partial restart occurs and q the probability for an appropriate crossover operation creating the global optimum after such partial restart. For the age-based and pessimistic value-based variant we see that it suffices to prove that $p^{-1}q^{-1} = O(\mu)$ holds, whereas for the optimistic value-based variant we need to show $p^{-1}q^{-1} = O(\mu + n/\log \mu)$.

Note that once we have at least two search points that are both locally optimal but have different age in the collection of search points this will always be the case until a restart happens. This is due to the smallest distance replacement where in case of equal fitness a search point with minimal age difference in selected for replacement. Hence, $p = p_1$ follows. Due to Lemma 10.7 the probability to have two locally optimal search points with different age when all search points have reached the local optimum is $p_1 = \Omega(1)$ for the first two variants and thus $p^{-1} = O(1)$. Due to Lemma 10.8 we have $p_1 = \Theta(\mu \log(\mu)/n)$ for sufficiently small $\mu = O(n/\log n)$ and $p_1 = \Theta(1)$ otherwise for the latter variant, leading to $p^{-1} = O(1 + n/\mu \log \mu)$.

We still need to derive the probability q for the different variants. Consider the point of time when the age of x, the first search point that has reached the local optimum, exceeds $\tau_{\max}$. In this iteration x and all its copies with identical age are removed and replaced by purely random search points. The expected takeover time for x to take over the complete collection of search points is $O(\mu \log \mu)$. If there are other points in the local optimum (with age different from x) the time until all search points are locally optimal can only be smaller. Since $\tau_{\max} = \omega(\mu n \log n)$ holds we have with probability close to 1 that all other search points are also locally optimal. Thus, after removing b copies of x we have a collection of search points with $\mu - b$ local optima and b random search points. Remember that $0 < b < \mu$ holds since we have a partial restart. Thus, with probability

$$q = \Omega\left(\frac{b}{\mu} \cdot \frac{\mu - b}{\mu}\right) = \Omega\left(\frac{1}{\mu}\right)$$

the global optimum is produced as next offspring. This establishes that on average $q^{-1} = O(\mu)$ such partial restarts suffice. Putting these things together, we get the claimed upper bound. □

Theorem 10.10 (Jansen and Zarges (2011c))**.** *Let $\mu \in \mathbb{N}$ with $\mu \geq 2$ and $\mu = n^{O(1)}$, p_c with $0 < \varepsilon \leq p_c \leq 1 - \varepsilon < 1$ for a positive constant ε, $k = O(1)$, and $\tau_{\max} = 2^{O(n)}$.*

The expected optimization time of A_{spa} using smallest age distance replacement from Definition 10.2 on f (Definition 10.3) is

$$E\left(T_{A_{spa},f}\right) = \Omega\left(\tau_{\max} + n^2 + \mu n \log n\right)$$

for age-based and pessimistic value-based static pure aging and

$$E\left(T_{A_{spa},f}\right) = \Omega\left(\left(1 + \frac{n}{\mu \log \mu}\right) \cdot \left(\tau_{\max} + n^2 + \mu n \log n\right)\right)$$

for optimistic value-based static pure aging (Definition 10.1).

Proof. From Lemma 10.6 we know that $\mathrm{E}\left(T_{A_{\mathrm{spa}},f}\right) = \Omega\left(p^{-1} q^{-1} \left(\tau_{\max} + n^2 + \mu n \log n\right)\right)$ holds where p is the probability that a partial restart occurs and q the probability for an appropriate crossover operation creating the global optimum after such partial restart.

Clearly, we need at least one successful partial restart to obtain the global optimum. Thus, we have the trivial lower bound for the age-based and pessimistic value-based variant following directly from Lemma 10.6. For the optimistic value-based variant we additionally know $p^{-1} = \Omega(1 + n/\mu \log \mu)$ due to Lemma 10.8, concluding the proof. □

We see that in the case of smallest age distance replacement the gap between the lower and the upper bound on the expected optimization time is $\Theta(\mu)$. This stems from the fact that we bounded the probability q for creating a globally optimal search points by means of crossover after a partial restart by $q = \Omega(1/\mu)$ and $q = O(1)$ respectively.

The smallest distance replacement has the property that the initial age structure of the collection of search points in the local optimum is not changed after the last search point enters the local optimum. This is due to the fact that after that point of time no new age value can enter the collection of search points as in the case of an non-improving iteration the age is always inherited of one of the parents. Hence, there is always at least one search point in the collection of search points that has the same age as the new search point. Since the age distance to these points is zero, simply two search points with the same age are exchanged. This is different for the most frequent replacement as shown in the next subsection.

10.5. Most Frequent Replacement Strategy

The most frequent replacement is probably the easiest and most direct way of preserving some degree of diversity with respect to age. Like smallest distance replacement it is effective enough to yield efficient optimization since again once we have at least two search points that are both locally optimal but have different age in the collection of search points this will always be the case until some restarts happens. Thus, the upper and lower bounds for smallest age distance replacement simply carry over to the most frequent replacement strategy as stated in the following two corollaries.

Corollary 10.11 (Jansen and Zarges (2011c)). *Let $\mu \in \mathbb{N}$ with $\mu \geq 2$ and $\mu = n^{O(1)}$, p_c with $0 < \varepsilon \leq p_c \leq 1 - \varepsilon < 1$ for a positive constant ε, $k = O(1)$, and $\tau_{\max} = \omega(\mu n \log \mu)$.*

The expected optimization time of A_{spa} using most frequent replacement from Definition 10.2 on f (Definition 10.3) is

$$E\left(T_{A_{spa},f}\right) = O\big(\mu \cdot \left(\tau_{\max} + n^2 + \mu n \log n\right)\big)$$

for age-based and pessimistic value-based static pure aging and

$$E\left(T_{A_{spa},f}\right) = O\left(\left(\mu + \frac{n}{\log \mu}\right) \cdot \left(\tau_{\max} + n^2 + \mu n \log n\right)\right)$$

for optimistic value-based static pure aging (Definition 10.1).

Corollary 10.12 (Jansen and Zarges (2011c)). *Let $\mu \in \mathbb{N}$ with $\mu \geq 2$ and $\mu = n^{O(1)}$, p_c with $0 < \varepsilon \leq p_c \leq 1 - \varepsilon < 1$ for a positive constant ε, $k = O(1)$, and $\tau_{\max} = 2^{O(n)}$.*

The expected optimization time of A_{spa} using most frequent replacement from Definition 10.2 on f (Definition 10.3) is

$$E\left(T_{A_{spa},f}\right) = \Omega\left(\tau_{\max} + n^2 + \mu n \log n\right)$$

for age-based and pessimistic value-based static pure aging and

$$E\left(T_{A_{spa},f}\right) = \Omega\left(\left(1 + \frac{n}{\mu \log \mu}\right) \cdot \left(\tau_{\max} + n^2 + \mu n \log n\right)\right)$$

for optimistic value-based static pure aging (Definition 10.1).

In contrast to smallest age distance replacement most frequent replacement changes the initial distribution of the different age values. It aims at obtaining and preserving a completely balanced distribution of ages within the collection of search points. Given such a balanced distribution better bounds on the expected optimization time can be proved. We consider Lemma 10.5 and remember that p denotes the probability for a partial restart and q denotes the probability that this partial restart is successful, i. e., generates a globally optimal search point. Now we replace p and q by p' and q' where p' denotes the probability to have a collection of search points completely within the local optimum with $r+1$ different ages. In this situation we will have r partial restarts within the next $\tau_{\max}$ steps or the global optimum is found. If q' denotes the probability that at least one of these partial restarts leads to the global optimum we obtain essentially the same bound as in Lemma 10.5. We state this as a corollary.

Corollary 10.13 (Jansen and Zarges (2011c)). *Let $\mu \in \mathbb{N}$ with $\mu \geq 2$ and $\mu = n^{O(1)}$, p_c with $0 < \varepsilon \leq p_c \leq 1-\varepsilon < 1$ for a positive constant ε, $k = O(1)$, and $\tau_{\max} = \omega(\mu n \log \mu)$. Moreover, let p' denote the probability that within the next $\tau_{\max}$ steps r partial restarts occur and q' the probability for an appropriate crossover operation creating the global optimum after one of these partial restarts.*

The expected optimization time of A_{spa} using an arbitrary strategy for static pure aging from Definition 10.1 and an arbitrary strategy for selection for replacement from Definition 10.2 on f (Definition 10.3) is

$$E\left(T_{A_{spa},f}\right) = O\left((p' \cdot q')^{-1}\left(\tau_{\max} + n^2 + \mu n \log n\right)\right).$$

Now we consider the different static pure aging strategies from Definition 10.1. The main difference between optimistic value-based aging and the two other variants (pessimistic value-based aging and age-based aging) is that in optimistic value-based aging a new age can only be introduced to the local optimum via mutation. Crossover operations without mutation need to involve one search point that already is a local optimum and one other search point. This other search point is worse with respect to function value in comparison to the other search point. Thus, in the pessimistic value-based variant this age is used and introduced as a (potentially) new age in the local optimum. Moreover, this other search point may be older than the search point that first entered the local optimum since this search point was assigned age 0 when it was created (as it was an improvement) and is thus younger than the other search points. If it is older, in the age-based variant again this age is used and introduced as (potentially) new age. In the optimistic value-based variant, however, the age of the better search points, i. e., the local optimum, is used and therefore the number of ages in the local optimum cannot increase. This does not only limit the number of different ages in the local optimum (in expectation it is $O(\mu \log(\mu)/n)$) but also ensures that the first point that entered the local optimum has maximal age in the local optimum. This yields a lower bound on the number of steps before a partial restart occurs that we can exploit to prove a better upper bound on the expected optimization time.

The following lemma makes a strong assumption on the age distribution at the local optimum. Given this assumption we can prove a high probability for finding a global optimum. We discuss afterwards how this assumption can be met.

Lemma 10.14 (Jansen and Zarges (2011c)). *Let $\mu \in \mathbb{N}$ with $\mu \geq 2$ and $\mu = n^{O(1)}$, p_c with $0 < \varepsilon \leq p_c \leq 1 - \varepsilon < 1$ for a positive constant ε, $k = O(1)$, and $\tau_{\max} = \omega(\mu n \log \mu)$. Consider A_{spa} using an arbitrary strategy for static pure aging from Definition 10.1 and an arbitrary strategy for selection for replacement from Definition 10.2 on f (Definition 10.3).*

Assume that the complete collection of search points is at the local optimum, i. e., $\{f(x) \mid x \in C\} = \{(5/4)n\}$. Let $r+1$ denote the number of different ages in C. For each of the $r+1$ different ages, let the number of $x \in C$ with each age be $\Theta(\mu/r)$ for the subsequent $\tau_{\max}$ steps or until a global optimum is found.

The probability that within the subsequent $\tau_{\max}$ steps a global optimum is found is $\Omega(1)$.

Proof. Due to our assumptions there are either r partial restarts in the subsequent $\tau_{\max}$ steps or the global optimum is found. For each of these restarts we have probability

$$\Theta\left(\frac{\mu/(r+1)}{\mu} \cdot \frac{\mu - \mu/(r+1)}{\mu}\right) = \Theta\left(\frac{1}{r+1} \cdot \left(1 - \frac{1}{r+1}\right)\right) = \Theta\left(\frac{1}{r}\right)$$

to select one locally optimal search point and one search point generated uniformly at random by the partial restart for crossover. A crossover of these points creates a global optimum with probability $\Omega(1)$ (Lemma 10.4) so that each of the r partial restarts has success probability $\Omega(1/r)$. The probability to have at least one of these successful is

$$1 - \left(1 - \Omega\left(\frac{1}{r}\right)\right)^r = \Omega(1)$$

as claimed. □

Note that Lemma 10.14 holds for all variants of A_{spa}. However, the assumption to always have $\Theta(\mu/r)$ search points for each of the r different ages is not realistic for all variants. But, for optimistic value-based aging in combination with the most frequent replacement strategy it is as the following lemma shows.

Lemma 10.15 (Jansen and Zarges (2011c)). *Let $\mu \in \mathbb{N}$ with $\mu \geq 2$ and $\mu = n^{O(1)}$, p_c with $0 < \varepsilon \leq p_c \leq 1 - \varepsilon < 1$ for a positive constant ε, $k = O(1)$, and $\tau_{\max} = \omega(\mu n \log \mu)$. Consider A_{spa} using optimistic value-based static pure aging from Definition 10.1 and the most frequent replacement strategy from Definition 10.2 on f (Definition 10.3).*

Moreover, consider the point of time when the number of different function values is reduced to 1 and all search points are in the local optimum, i. e., $\{f(x) \mid x \in C\} = \{(5/4)n\}$.

The conditions of Lemma 10.14 are met with probability $\Omega(1)$.

Proof. Consider the first search point x to enter the local optimum. It is assigned age 0 at this point of time and as argued above no search point with a larger age can enter the local optimum. Consider another search point z that enters the local optimum. Clearly, z was created either involving crossover or by mutation only. If it was created by mutation of a search point that is not locally optimal z is an improvement and $\text{age}.z = 0 < \text{age}.x$ holds. If z is a clone of a local optimum it inherits its age. Thus, by means of mutation no search point with larger age can be introduced into the local optimum. Now, consider the case where z is created by means of crossover. If both search points are not locally optimal again z is an improvement. If at least one search point is a local optimum than z inherits its age since we are using optimistic value-based aging. Thus, also crossover cannot introduce a search point with age larger than $\text{age}.x$ and hence, x has maximal age in the local optimum.

This yields, that the first (partial) restart after x entered the local optimum occurs after $\tau_{\max} = \omega(\mu n \log \mu)$ steps. The expected takeover time for the complete collection of search points is $\Theta(\mu \log \mu)$. Thus, on expectation after $O(\mu \log \mu)$ steps we have the complete collection of search points in the local optimum and this can only change when the first partial restart occurs, i. e., after $\tau_{\max} - O(\mu \log \mu) = \omega(\mu n \log \mu)$ steps. Thus, there is sufficient time to obtain a balanced distribution of the ages within the local optimum. $\square$

Lemma 10.15 proves $q' = \Omega(1)$. All we need to obtain a better upper bound on the expected optimization time is a bound on p'. We recall that we already have such a bound.

Theorem 10.16 (Jansen and Zarges (2011c))**.** *Let $\mu \in \mathbb{N}$ with $\mu \geq 2$ and $\mu = n^{O(1)}$, p_c with $0 < \varepsilon \leq p_c \leq 1 - \varepsilon < 1$ for a positive constant ε, $k = O(1)$, and $\tau_{\max} = \omega(\mu n \log \mu)$.*

The expected optimization time of A_{spa} using optimistic value-based static pure aging from Definition 10.2 and most frequent replacement from Definition 10.2 on f (Definition 10.3) is

$$E\left(T_{A_{spa},f}\right) = \Theta\left(\left(1 + \frac{n}{\mu \log \mu}\right) \cdot \left(\tau_{\max} + n^2 + \mu n \log n\right)\right).$$

Proof. We apply Corollary 10.13. According to Lemma 10.14 and Lemma 10.15 we have $q' = \Omega(1)$. Moreover, Lemma 10.8 yields $p' = \Omega(\mu \log(\mu)/n)$ for not too large μ and $p' = \Omega(1)$ otherwise. Together this yields the claimed upper bound. The lower bound is already contained in Corollary 10.12. $\square$

Unfortunately, we are not able to prove a similar result for the other two variants of static pure aging. Since older search points can enter the local optimum we have no lower bound on the number of steps until a partial restart occurs. This implies that we cannot prove that the age structure is balanced and thus are unable to prove that the conditions of Lemma 10.14 are met. The improvement of the upper bounds to obtain tight bounds for these static pure aging variants is an open problem.

10.6. Fewest Replacement Strategy

In contrast to the former two replacement strategies, fewest replacement does not incorporate any age diversity mechanism. Even worse, diversity with respect to age is intentionally destroyed. In the next theorem we show that such an algorithm is not able to optimize our considered example function with overwhelming probability even in an exponential number of steps.

Theorem 10.17. *Let $\mu \in \mathbb{N}$ with $\mu \geq 2$ and $\mu = n^{O(1)}$, p_c with $0 < \varepsilon \leq p_c \leq 1 - \varepsilon < 1$ for a positive constant ε, $k = O(1)$, and $\tau_{\max} = \omega(\mu n \log \mu)$.*

The optimization time of A_{spa} using fewest replacement from Definition 10.2 and optimistic value-based aging from Definition 10.1 on f (Definition 10.3) is

$$T_{A_{spa},f} \geq 2^{cn}$$

($c > 0$ a sufficiently small constant) with probability $1 - 2^{-\Omega(n)}$.

Proof. Consider the first point in time when all search points are locally optimal. As seen in the proof of Lemma 10.15 the age of all search points is $O(\mu \log \mu)$ at this point in time. With probability $1 - 2^{-\Omega(n)}$ all ages are bounded by $O(\mu n \log \mu)$. Thus, there are still $\tau_{\max} - O(\mu n \log \mu) = \omega(\mu n \log \mu)$ steps left before the first (partial) restart can occur.

Let x be a search point with some age x.age that occurs most frequently in the current collection of search points. If there exist more than one such age we select an arbitrary one. Due to the replacement strategy the number of search points with that age will not decrease during the ongoing search process. Moreover, any time such a point is selected and copied the number of search points with the same age increases and the number of search points with other ages decreases. Thus, after expected $\Theta(\mu \log \mu)$, say $d\mu \log \mu$ ($d > 0$ constant), steps the collection of search points only consists of search points with only one single age, keeping the algorithm from performing a partial restart and yielding inefficient optimization time. Due to Markov's inequality the probability not to have such an event within $2d\mu \log \mu$ is at most $1/2$. Moreover, the probability not to have such an event in n rounds of $2d\mu \log \mu$ steps can be bounded above by $(1/2)^n$. Since $\tau_{\max} = \omega(\mu n \log \mu)$ this yields the theorem. □

The proof of Theorem 10.17 does not work for age-based and pessimistic value-based static pure aging. These two different aging variants allow that search points may enter the local optimum that are older than the first search point that entered the local optimum. Since the age of these search points may be much older it cannot be ruled out that these search point cause a partial restart rather quickly and thus lead to successful optimization. However, it seems to be highly unlikely that very old search points survive long enough for this event to happen. We therefore speculate that Theorem 10.17 can be generalized for the other two aging variants, too. This, however, is currently an open problem.

10.7. Random Replacement Strategy

Finally we consider the random replacement variant. Here again no diversity mechanism with respect to age is used but in contrast to the fewest replacement diversity it is just not cared about. Thus, we simply replace a random search point with worst fitness value. This is equivalent to the standard replacement strategy in randomized search heuristics where age is not used at all. We prove that this also leads to inefficient optimization time if used in combination with the optimistic value-based aging strategy. We speculate that the same holds for the other aging strategies and discuss difficulties in proving this after the proof of the following result. This results demonstrates that age diversity mechanisms are an important concept for effective aging operators.

Theorem 10.18 (Jansen and Zarges (2011c))**.** *Let $\mu \in \mathbb{N}$ with $\mu \geq 2$ and $\mu = n^{O(1)}$, p_c with $0 < \varepsilon \leq p_c \leq 1 - \varepsilon < 1$ for a positive constant ε, $k = O(1)$, and $\tau_{\max} = \omega(\mu n \log \mu)$.*

The optimization time of A_{spa} using random replacement from Definition 10.2 and optimistic value-based aging from Definition 10.1 on f (Definition 10.3) is

$$T_{A_{spa},f} \geq 2^{cn}$$

($c > 0$ a sufficiently small constant) with probability $1 - 2^{-\Omega((n \log\log \mu)/\log \mu)}$.

Proof. We aim at proving that with probability close to 1 no partial restart will occur. This immediately implies the result since such a partial restart is needed for efficient optimization of f.

We consider the situation when $\max\{f(x) \mid x \in C\}$ is increased to $(5/4)n$, i. e., a first point x enters the local optimum. Since this point is an improvement it is assigned age 0. We consider the subsequent steps and are interested in the first point of time when $\min\{f(x) \mid x \in C\} = \{(5/4)n\}$ holds, i. e., the complete collection of search points is in the local optimum. As seen in the proof of Theorem 10.15, at this point of time x (and its descendants) have maximal age in C.

We claim that we have $x.\text{age} = O\big(\mu \log^2 \mu\big)$ at this point of time with probability $1 - 2^{-\Omega\left(\log^2 \mu\right)}$. As noted before the process of taking over the collection of search points is similar to the coupon collector process yielding an expected duration of $O(\mu \log \mu)$. The bound $\mu^{-\Omega(\beta)}$ on the probability for taking $\Omega(\beta\mu \log \mu)$ steps (Lemma B.17) carries over, too. Setting $\beta = \log \mu$ we obtain the bound $2^{-\Omega\left(\log^2 \mu\right)}$ as claimed. Note that during this process in all steps the order of growth of the probability of inserting another point to the local optimum is bounded above by the probability of adding another copy of x to the local optimum. Consequently, also with probability $1 - 2^{-\Omega\left(\log^2 \mu\right)}$ we have $\Omega(\mu)$ copies of x in C. This implies that in each selection such a search point is selected with probability $\Omega(1)$.

We consider the subsequent steps and claim that after some number of steps (where the number of steps will be discussed afterwards) x will have taken over the complete collection of search points (and thus there is only one age present in C) with probability $1 - 2^{-\Omega(\mu)}$. Note that if this happens before a partial restart happens no partial restart can happen anymore. Let n_x denote the number of copies of x in the beginning. Remember that we have $n_x = \Omega(\mu)$ with probability close to 1. In one round this number n_x may remain unchanged or it may either by increased or decreased by exactly 1. We want to prove that it will be increased to μ with probability close to 1.

Since we consider the situation when the whole collection of search points is in the local optimum we have that all search points have equal fitness and thus the age of the new search points equals $\max\{\text{age}.x, \text{age}.y\}$. Moreover, crossover of any two parents can only yield another local optimum as result. The event 'n_x is increased' can happen with and without crossover. Without crossover it happens if one such search point is selected (probability n_x/μ), it is not changed by mutation (probability $(1 - 1/n)^n$), and none of the n_x search points is selected for replacement (probability $(\mu - n_x)/\mu$). This leads to a contribution of $(1 - p_c)(n_x/\mu)(1 - 1/n)^n(\mu - n_x)/\mu$ to $\text{Prob}\,(n_x \text{ is increased})$ by this case. With crossover it happens if at least one such search point is selected (probability $1-((\mu-n_x)/\mu)^2 = (n_x/\mu)(2-n_x/\mu))$, the result of crossover is not changed by mutation (probability $(1 - 1/n)^n$), and none of the n_x search points is selected for replacement (probability $(\mu - n_x)/\mu$). Together we obtain

$$
\begin{aligned}
&\text{Prob}\,(n_x \text{ is increased})\\
&\quad= (1-p_c)\frac{n_x}{\mu}\left(1-\frac{1}{n}\right)^n\frac{\mu-n_x}{\mu} + p_c\frac{n_x}{\mu}\left(2-\frac{n_x}{\mu}\right)\left(1-\frac{1}{n}\right)^n\frac{\mu-n_x}{\mu}\\
&\quad= \frac{n_x(\mu-n_x)}{\mu^2}\left(1-\frac{1}{n}\right)^n\left((1-p_c)+p_c\cdot\left(2-\frac{n_x}{\mu}\right)\right)\\
&\quad= \frac{n_x(\mu-n_x)}{\mu^2}\left(1-\frac{1}{n}\right)^n\left(1+p_c-p_c\frac{n_x}{\mu}\right).
\end{aligned}
$$

The event 'n_x is decreased' can also happen with and without crossover. Without crossover it happens if some other search point is selected (probability $(\mu - n_x)/\mu$), it is not changed by mutation (probability $(1 - 1/n)^n$), and one of the n_x search points is selected for replacement (probability n_x/μ). This leads to a contribution of $(1 - p_c)$ $\cdot((\mu - n_x)/\mu)(1 - 1/n)^n n_x/\mu$ to $\text{Prob}\,(n_x \text{ is decreased})$ by this case. With crossover it happens if no such search point is selected (probability $((\mu - n_x)/\mu)^2$), the result of crossover is not changed by mutation (probability $(1 - 1/n)^n$), and one of the n_x search points is selected for replacement (probability n_x/μ). Together we obtain

$$
\begin{aligned}
&\text{Prob}\,(n_x \text{ is decreased})\\
&\quad= (1-p_c)\frac{\mu-n_x}{\mu}\left(1-\frac{1}{n}\right)^n\frac{n_x}{\mu} + p_c\left(\frac{\mu-n_x}{\mu}\right)^2\left(1-\frac{1}{n}\right)^n\frac{n_x}{\mu}\\
&\quad= \frac{n_x(\mu-n_x)}{\mu^2}\left(1-\frac{1}{n}\right)^n\left(1-p_c+p_c\frac{\mu-n_x}{\mu}\right)
\end{aligned}
$$

hold. For the sake of comparison we consider

$$\frac{\text{Prob}\,(n_x \text{ is increased})}{\text{Prob}\,(n_x \text{ is decreased})} = \frac{1 + p_c - p_c n_x/\mu}{1 - p_c + p_c(\mu - n_x)/\mu}$$
$$= \frac{1 + p_c - p_c n_x/\mu}{1 - p_c \cdot n_x/\mu} = 1 + \frac{p_c}{1 - p_c \cdot n_x/\mu} > 1 + p_c$$

and see a clear tendency towards increasing n_x. For the analysis we consider a Markov chain $X_0, X_1, \ldots$ on the state space $\{0, 1, \ldots, \mu\}$ with $\text{Prob}\,(X_{t+1} = 0) = 1$ for $X_t = 0$, $\text{Prob}\,(X_{t+1} = \mu) = 1$ for $X_t = \mu$, $\text{Prob}\,(X_{t+1} = X_t + 1) = (1 + p_c)/(2 + p_c)$, and $\text{Prob}\,(X_{t+1} = X_t - 1) = 1/(2 + p_c)$ in all other cases. Clearly, all transition probabilities not explicitly stated are 0. This Markov chain corresponds to the algorithm conditioned on the event that n_x changes and is pessimistic with respect to having $n_x = \mu$ at some point of time. Moreover, the Markov chain corresponds exactly to the situation in the gambler's ruin theorem (Lemma B.18). Remember that we have $n_x = \Omega(\mu)$, say $n_x = c\mu$, initially. Thus, the probability not to have $n_x = \mu$ at some point of time is bounded above by

$$\frac{(1 + p_c)^{c\mu} - 1}{(1 + p_c)^{\mu} - 1} = \left(\frac{1}{1 + p_c}\right)^{(1-c)\mu} \cdot \frac{(1 + p_c)^{\mu} - (1 - p_c)^{(1-c)\mu}}{(1 + p_c)^{\mu} - 1}$$
$$= \left(\frac{1}{1 + p_c}\right)^{(1-c)\mu} \cdot \left(1 - \frac{(1 + p_c)^{(1-c)\mu} - 1}{(1 + p_c)^{\mu} - 1}\right) = 2^{-\Omega(\mu)}.$$

The expected duration of the random process described by the Markov chain is $O(\mu)$. However, this is different from the duration of the random process in the algorithm since we considered the situation conditioned that n_x is changed. Thus, we need to take into account the probability to change n_x in one step. This probability is given by

$$\text{Prob}\,(n_x \text{ is increased}) + \text{Prob}\,(n_x \text{ is decreased}) = \Omega\!\left(\frac{n_x(\mu - n_x)}{\mu^2}\right)$$

and we see that it is particularly small when $n_x = O(1)$ or $\mu - n_x = O(1)$ holds. We improve the trivial bound $O(\mu^2)$ on the duration to $O\big((\mu \log^2 \mu) / \log\log \mu\big)$ in the following way.

Consider the situation with $n_x = O(1)$ for $\Theta(\mu \log \mu)$ steps. Since the probability to increase n_x by 1 is bounded below by $\Omega(1/\mu)$ in this situation we have on expectation $n_x = \Omega(\log \mu)$ after these steps. Consider another round of $\Theta(\mu \log \mu)$ steps. Now the probability to increase n_x is bounded below by $\Omega(\log(\mu)/\mu)$ and we expect to have $n_x = \Omega(\log^2 \mu)$ at the end. In general, after r such rounds we expect $n_x = \Omega(\log^r \mu)$. Thus, after $\Theta(\log(\mu)/\log\log \mu)$ rounds we have $n_x = \Omega\Big(\log^{\log(\mu)/\log\log \mu} \mu\Big) = \Omega(\mu)$ in the end. For $\mu - n_x$ the situation is symmetric (but reversed in time). Thus, we have an expected length of $O\big((\mu \log^2 \mu) / \log\log \mu\big)$ as claimed.

In summation we have that on expectation after $O\big((\mu \log^2 \mu) / \log\log \mu\big)$ steps the complete collection of search points is of the same age so that a partial restart is impossible. To obtain the result with probability very close to 1 we consider $\Theta(\mu n \log \mu)$ steps in total. These steps can be considered as $\Theta((n \log \log \mu)/ \log \mu)$ repetitions of length $\Theta\left(\left(\mu \log^2 \mu\right) / \log \log \mu\right)$ each. This yields the desired bound on the probability. □

We speculate that a similar result holds for age-based and pessimistic value-based aging. However, proving this is an open problem. The difficulty is essentially the same as for Theorem 10.17.

10.8. Experimental Supplements

The results presented in the preceding sections give insights into what aging can achieve in randomized search heuristics. Our theoretical analyses give a coarse picture of the effects of aging, in particular with respect to partial restarts. Nevertheless, not all questions are answered. First of all, most of the derived bounds are not tight. Second, our results are asymptotic in nature yielding to the same drawbacks as discussed previously.

It is not obvious what good values for μ are. However, the theoretical results give hints. The bounds for the pessimistic value-based variant and the age-based variant indicate that a smaller size of the collection of search points leads to a smaller optimization time. For the optimistic value-based variant, the bounds suggest $\mu = \Theta(n/ \log n)$ as a good choice. However, as our theoretical bounds are not tight, these speculations may be wrong.

We do all experiments with sizes $\mu \in \{2, \lfloor \sqrt{n} \rfloor, \lfloor n/ \log n \rfloor, n\}$ for the collection of search points. Clearly, $\mu = 2$ is interesting as it is the smallest possible size and possibly a good choice for the pessimistic value-based and the age-based variant. For the same reason we pick $\mu = \lfloor n/ \log n \rfloor$ for the optimistic value-based variant. The choice $\mu \approx \sqrt{n}$ has often turned out to be a good choice (Harik et al. 1999) and was also studied in the previous sections. Moreover we are interested in the effects of sizes that are not sub-linear. Thus, we also pick $\mu = n$.

We require $\tau_{\max} = \omega(\mu n \log \mu)$ and choose $\tau_{\max} = \lfloor 6 \mu n \log(\mu) \log n \rfloor$ for our experiments where again the factor 6 helps for small values of n. All bounds work for arbitrary constant crossover probabilities p_c. We use $p_c = 0.5$, a medium sized value. All proofs work for any constant number k of crossover points. We consider the commonly used 1-point crossover.

We perform two sets of experiments. First of all, we consider the optimization times for the most frequent replacement and all static pure aging variants. Second, we compare the optimization times of the other replacement strategies with most frequent replacement. The results of the different experiments are given in the following subsections.

10.8.1. Optimization Times of the Most Frequent Replacement Strategy

We analyze the optimization times of most frequent replacement combined with the three static pure aging strategies and different values for the size μ of the collection of search points. Table 10.1 shows the resulting bounds on the expected optimization times for this setting due to Corollary 10.12 for the lower bounds, Corollary 10.11 for the upper bound of age-based and pessimistic value-based aging and Theorem 10.16 for the

	age-, pessimistic value-based	optimistic value-based
$\mu = 2$	$\Theta(n^2)$	$\Theta(n^3)$
$\mu = \lfloor\sqrt{n}\rfloor$	$\Omega(n^2)$, $O(n^{5/2})$	$\Theta(n^{5/2}/\log n)$
$\mu = \lfloor n/\log n\rfloor$	$\Omega(n^2 \log n)$, $O(n^3)$	$\Theta(n^2 \log n)$
$\mu = n$	$\Omega(n^2 \log^2 n)$, $O(n^3 \log^2 n)$	$\Theta(n^2 \log^2 n)$

Table 10.1.: Bounds on the expected optimization time for example sizes of the collection of search points using the most frequent replacement strategy.

improved upper bound of optimistic value-based aging. Note, that we also inserted the concrete value for $\tau_{\max}$ that is used within the experiments when deriving these bounds. We see that for the considered parameter settings we have a gap of $\Theta(\mu)$ for age-based and pessimistic value-based aging while for the optimistic value-based we have a tight result.

As done in previous experimental analyses, we perform 100 independent runs for each setting and plot the results using box-and-whisker plots (Definition B.1) for $n \in \{20, 40, \ldots, 1000\}$. Due to the excessive computation time, we only consider $n \in \{20, 40, \ldots, 340\}$ for $\mu = n$ (all variants) and $n \in \{20, 40, \ldots, 460\}$ for $\mu = 2$ (optimistic value-based). The results are shown in Figure 10.3 for age-based aging, in Figure 10.4 for optimistic value-based aging and in Figure 10.5 for pessimistic value-based aging where the number of iterations are drawn in logarithmic scale. To facilitate comparison we plot the medians for all sizes of the collection of search points in one joint diagram in each case (Figures 10.3-10.5, bottom) in linear scale. We additionally plot $c \cdot b(n)$ if we have a bound $\mathrm{E}\left(T_{A_{\text{spa}}}, f\right) = O(b(n))$ where c is determined via a least squares fit.

It is obvious that for most frequent replacement the variance decreases with increasing size of the collection of search points. This is due to the fact that the probability for a partial restart increases with increasing μ. Consider the situation just after the first search point reached the local optimum. In the extreme case $\mu = 2$ there is only one other search point left that needs to enter the local optimum with a different age in order to allow for a partial restart. For larger μ more trials are possible. If the algorithm fails to perform a partial restart, a complete restart is required. Certainly, complete restarts are rather expensive and lead to larger variances in the optimization time. For $\mu = n$ we see that generally no complete restarts occur during the optimization process and we succeed in performing a suitable crossover operation when we reach the local optimum for the first time.

Moreover, we see that the variance is larger for optimistic value-based aging than for the other two variants. This is due to the fact, that here crossover is not able to create a search point with a different age in the local optimum whereas this is possible for the other two aging variants. Additionally, the variance for age-based aging is slightly larger than for pessimistic value-based aging. This can again be explained with the effects of the crossover operator: in pessimistic value-based aging always the age of the worse search point is inherited (if it is not an improvement). In case of a crossover of a locally

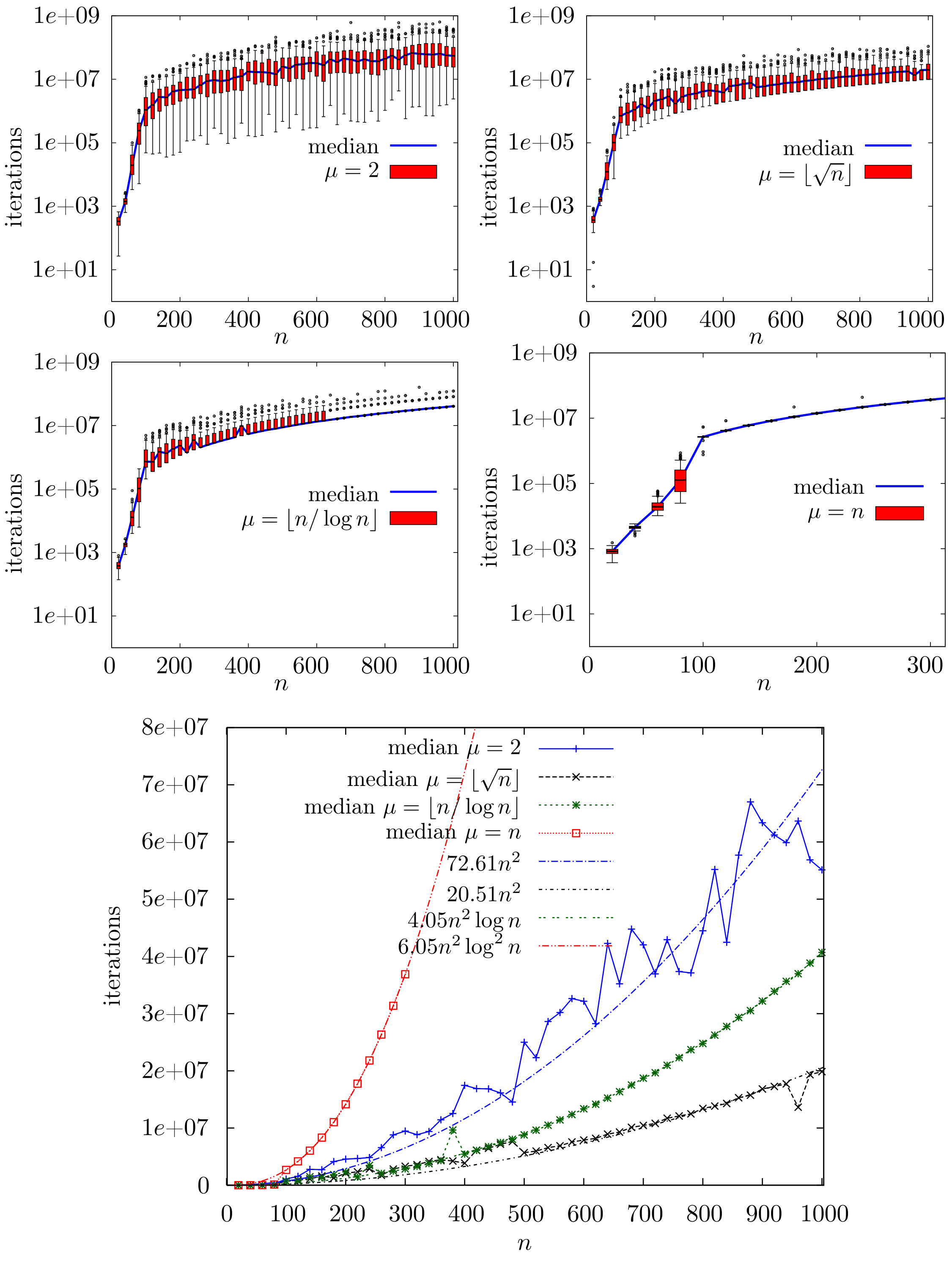


Figure 10.3.: Experimental results for most frequent replacement and age-based aging with different values for the size μ of the collection of search points, data from 100 runs.

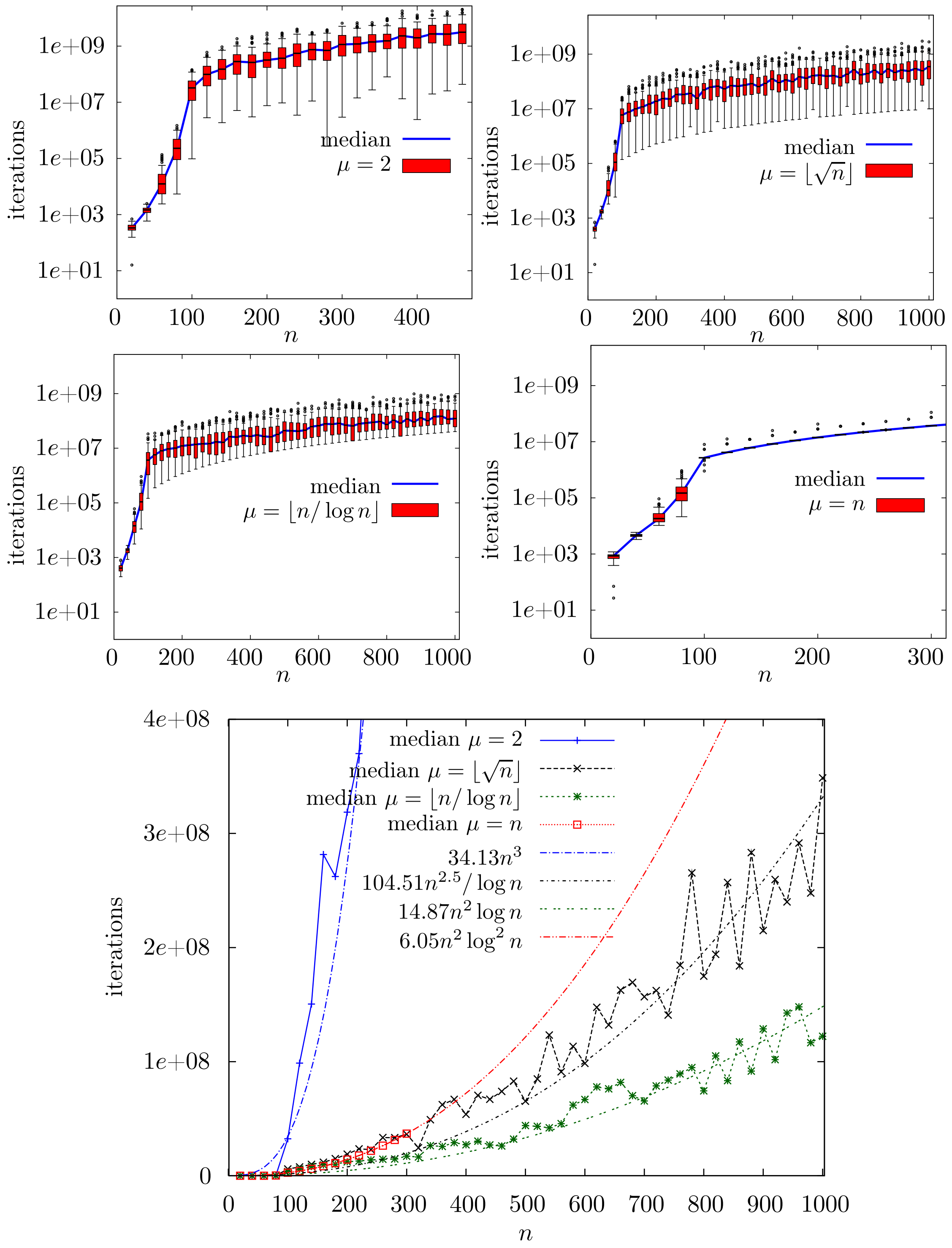


Figure 10.4.: Experimental results for most frequent replacement and optimistic value-based aging with different values for the size μ of the collection of search points, data from 100 runs.

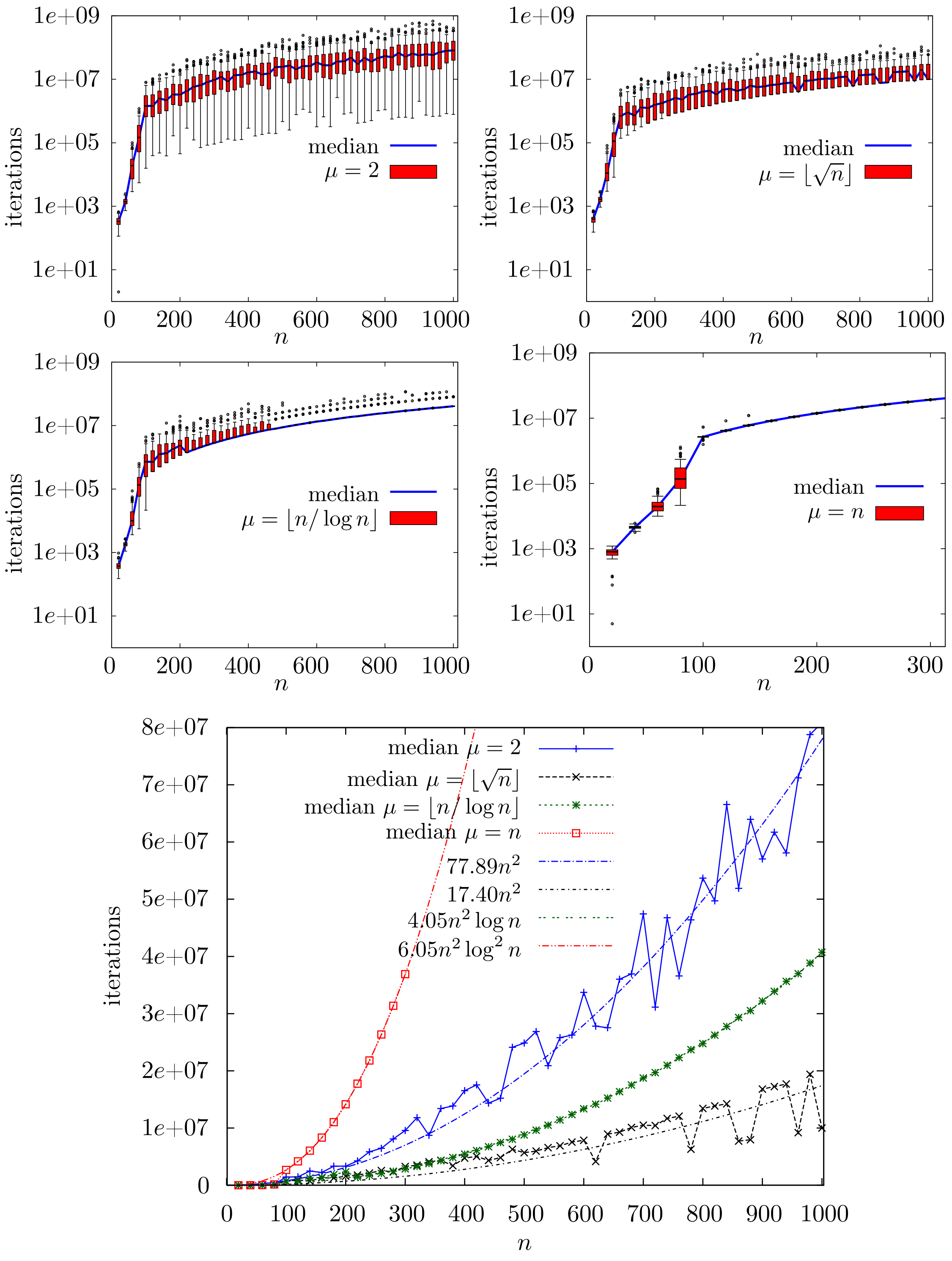


Figure 10.5.: Experimental results for most frequent replacement and pessimistic value-based aging with different values for the size μ of the collection of search points, data from 100 runs.

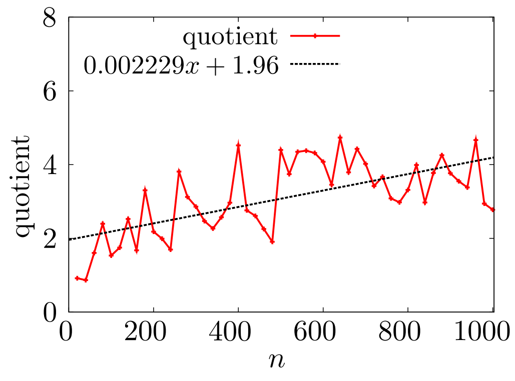
optimal search point x and a worse search point y that creates another locally optimal search point z, this means that another age is introduced if y.age was not already present in the local optimum. Since the first locally optimal search point is always created by means of mutation, this is always the case as long as there is only one age value in the local optimum. Thus, in this situation a suitable crossover operation always introduces a second age. This is not true for the age-based variant as additionally $y.\text{age} > x.\text{age}$ must hold. We conclude that pessimistic value-based aging is more robust than age-based aging with respect to introducing a second age and hence allowing for a partial restart whereas optimistic value-based aging is least robust in this respect.

We compare the effects of the size of the collection of search points (Figures 10.3-10.5, bottom). First, note that not only the theoretical upper and lower bounds for age-based and pessimistic value-based aging are identical but in fact there is hardly any difference visible in the experimental results. This lack of empirical difference is also present in the experimental results for $\mu = n$ in all three aging variants. For the optimistic value-based variant, we see that the experimental results are in good accordance with the theoretical results, namely $\mu = \lfloor n/\log n \rfloor$ being the fastest and $\mu = 2$ being by far the worst.

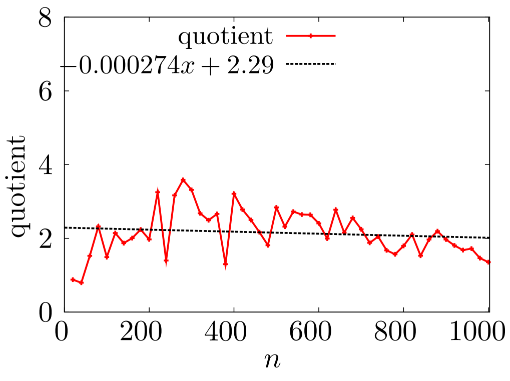
For the age-based and pessimistic value-based variant, surprisingly, the algorithm with size $\mu = 2$ is clearly outperformed by its counterparts with sizes $\mu = \lfloor \sqrt{n} \rfloor$ and $\mu = \lfloor n/\log n \rfloor$. This shows that (at least for not too large values of n) the asymptotic theoretical results are misleading from a practical point of view. This is due to the large constants hidden in the derived asymptotic bounds and thus, these bounds do not reflect the actual optimization times on the small input sizes considered in the experimental study. However, it is not clear how large n needs to be chosen in order to obtain results that are in correspondence with the asymptotic theoretical results.

We speculate that our upper bounds are not tight. We support this hypothesis by plotting the fitted lower bounds together with the empirical mean in Figures 10.3-10.5 (bottom) and find a good fit in all cases. That larger sizes of the collection of search points outperform the choice $\mu = 2$ at least partially contradicts the theoretical results. Thus, we take a closer look by comparing the quotients of the observed means.

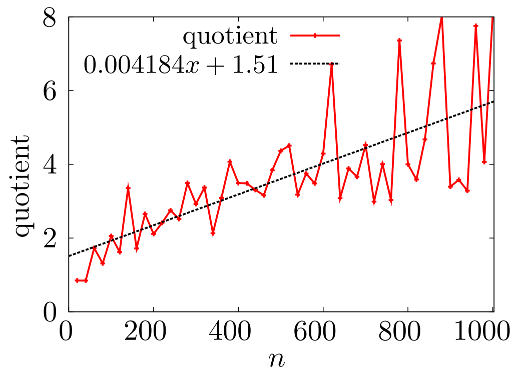
The theoretical bounds predict the quotient for $\mu = 2$ over $\mu = \lfloor n/\log n \rfloor$ to converge to 0. For $\mu = 2$ over $\mu = \lfloor \sqrt{n} \rfloor$ it should be bounded above by a positive constant. Note that the theoretical bounds are asymptotic and predict this behavior for $n \to \infty$. For $\mu = 2$ over $\mu = \lfloor n/\log n \rfloor$ (Figure 10.6(b) and Figure 10.6(d)) we see that after an increase for small values of n the quotient does indeed decrease. We fit the graph of the linear function $a \cdot x + b$ to the data and see that already for $n \leq 1000$ the results match the asymptotic bounds. Things are different for $\mu = 2$ over $\mu = \lfloor \sqrt{n} \rfloor$ (Figure 10.6(a) and Figure 10.6(c)). Instead of being obviously bounded the quotient increases. This impression is confirmed when fitting $a \cdot x + b$ to the data. It is impossible to say from these experiments if the values of n considered are still too small or if the high variance is to blame.



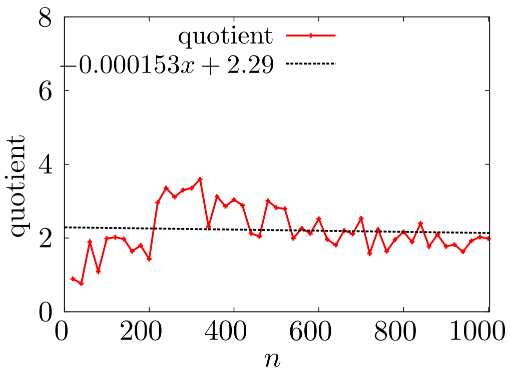
(a) $\mu = 2$ and $\mu = \lfloor\sqrt{n}\rfloor$, age-based



(b) $\mu = 2$ and $\mu = \lfloor n/\log n\rfloor$, age-based



(c) $\mu = 2$ and $\mu = \lfloor\sqrt{n}\rfloor$, pessimistic value-based



(d) $\mu = 2$ and $\mu = \lfloor n/\log n\rfloor$, pessimistic value-based

Figure 10.6.: Quotients of the observed medians, data from 100 runs.

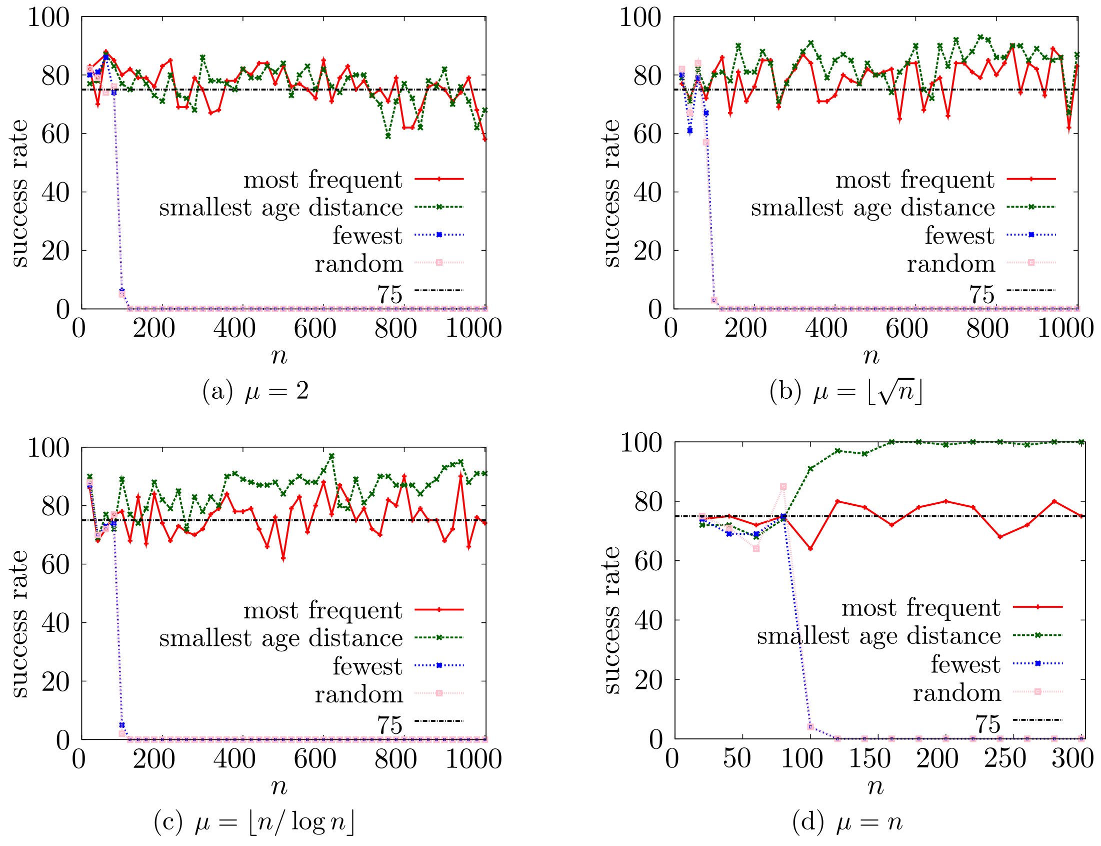
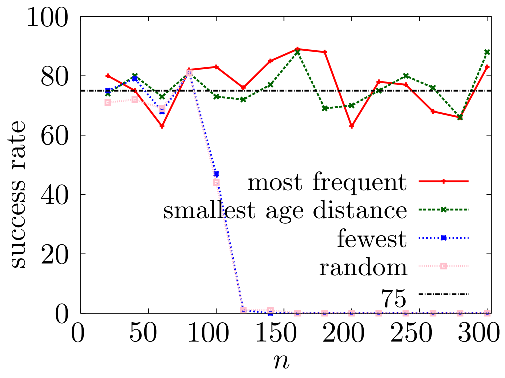


Figure 10.7.: Experimental results for the different replacement strategies and age-based aging, data from 100 runs.

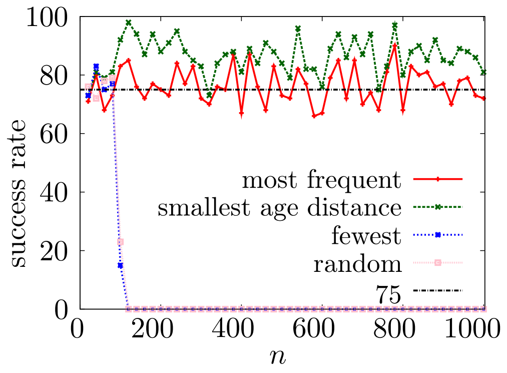
10.8.2. Comparison of the Different Replacement Strategies

In order to compare the four replacement strategies from Definition 10.2 we perform experiments with all four of them and the parameter settings from above. We fix the maximal number of iterations executed to the corresponding upper quartile from the first set of experiments. The results are given as a plot for each μ and aging variant showing the number of successful runs within 100 independent runs for considered values of n: Figure 10.7 for age-based aging, Figure 10.8 for optimistic value-based aging and Figure 10.9 for pessimistic value-based aging.

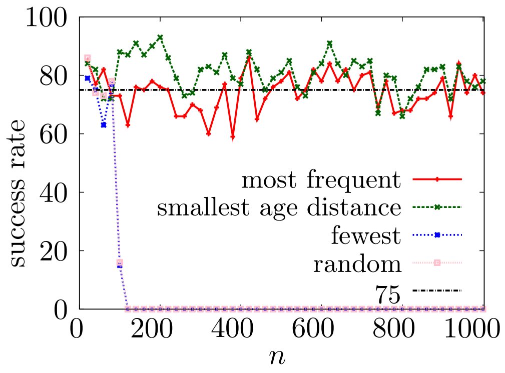
In all settings considered it becomes apparent that the success rate of random replacement and fewest replacement starts decreasing for $n \approx 100$ and then converges to 0 very quickly. We can conclude that already for quite small values of n these two strategies are ineffective since with high probability one single age takes over the collection of search points in the local optimum, preventing the algorithm from performing a partial restart. This can be observed for all three aging variants. Note, that we only derived bounds for these two replacement strategies in the optimistic value-based variant. However, the



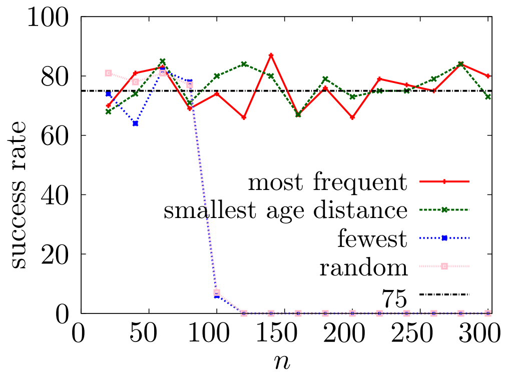
(a) $\mu = 2$



(b) $\mu = \lfloor \sqrt{n} \rfloor$

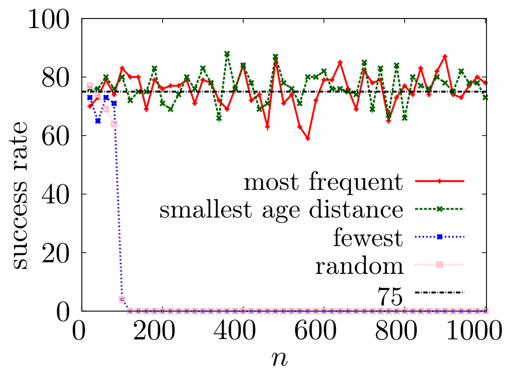


(c) $\mu = \lfloor n/\log n \rfloor$

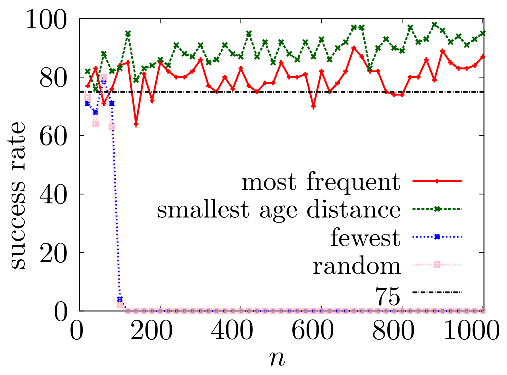


(d) $\mu = n$

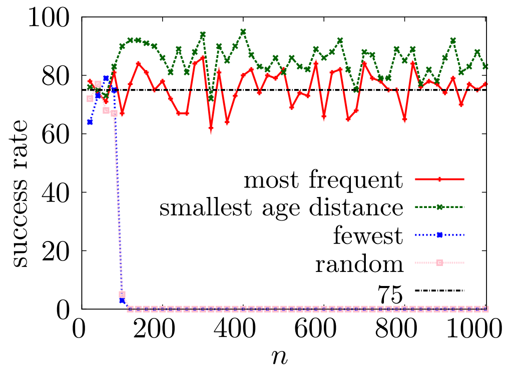
Figure 10.8.: Experimental results for the different replacement strategies and optimistic value-based aging, data from 100 runs.



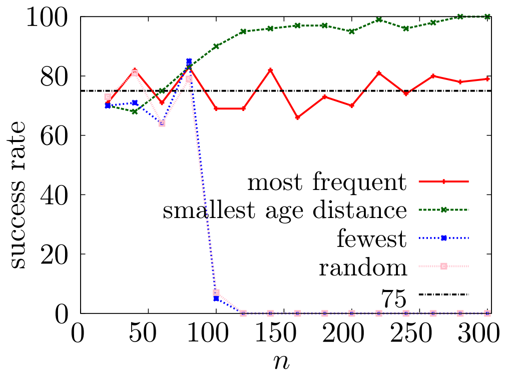
(a) $\mu = 2$



(b) $\mu = \lfloor\sqrt{n}\rfloor$



(c) $\mu = \lfloor n/\log n\rfloor$



(d) $\mu = n$

Figure 10.9.: Experimental results for the different replacement strategies and pessimistic value-based aging, data from 100 runs.

experimental results support our speculation that similar bounds hold for the other two aging strategies.

In contrast to that observation both, most frequent replacement and smallest age distance replacement, are effective. Note, that for most frequent replacement we expect a success rate of 75% since we use the upper quartile of the number of iterations during 100 independent runs of this algorithmic variant. Our expectations are met in all settings considered here. Surprisingly, the success rate of smallest age distance replacement is larger for increasing size of the collection of search points μ for the limited range of values inspected. This is in particular true for age-based and pessimistic value-based aging where for $\mu = n$ a success rate of nearly 100% is realized. Note, that the effect is already visible for $\mu = \lfloor\sqrt{n}\rfloor$ and $\mu = \lfloor n/\log n\rfloor$. For optimistic value-based aging the superiority of smallest age distance replacement is not that obvious. It appears that the success rate for $\mu = \lfloor\sqrt{n}\rfloor$ and $\mu = \lfloor n/\log n\rfloor$ is slightly higher but surprisingly it is not for $\mu = n$. Thus, it is not clear if these effects are only due to the random fluctuation.

The observations can be explained as follows. Since in optimistic value-based aging crossover does not help in creating different ages in the local optimum, we expect less different ages in the local optimum in comparison to the other two aging variants. Additionally, most frequent replacement tends to quickly equally distribute the quantities of the different age values. Having more age values, decreases the proportion of a single age value and decreases the probability to perform an appropriate crossover operation in a single restart. In smallest age distance replacement, the initial proportions of the different ages are not changed. It seems that for age-based and pessimistic value-based aging partial restarts and the effects of several successive partial restarts are more effective if the quantities of different age values are not equally distributed whereas for optimistic value-based aging both mechanisms yield very similar performance.

Part IV.

Bridging the Gap Between Theory and Practice

11. Introduction

In the preceding chapters of this thesis we have investigated different aspect of artificial immune systems. In particular we considered typical operators used in such algorithms. We derived rigorous, asymptotic results on the optimization time of these operators when embedded into a simple algorithmic framework, analyzing their performance in isolation as much as possible. Although, we have upper and lower bounds in many cases, these bounds are usually not tight. Moreover, asymptotic results may not describe the situation for typical problem dimensions, in particular small problem sizes. This is the reason why, whenever reasonable, we presented experimental supplements in order to investigate further aspects and address the practical relevance of our theoretical results. For example, in Chapter 4 we showed that in experiments the mutation operator from CLONALG (Definition 4.3) yields good results if n is not too large although we proved an exponential lower bound on the optimization time with high probability. In Chapter 10 we were able to demonstrate advantages of larger collection of search points although our theoretical bounds indicated that small collections of search points yield better performance. We investigated the role of deterministic initialization for contiguous hypermutations (Chapter 5), the robustness of an algorithm using aging with respect to the maximal lifespan (Chapter 8 and 9), and the effect of different replacement strategies combined with aging (Chapter 10). All these results gained more insights into the aspect considered in the theoretical parts of this thesis and are hoped to contribute to bridging the gap between theory and practice.

11.1. Related Work

Bridging the gap between theory and practice is an important and broad area of research. This is in particular true for randomized search heuristics since these kind of algorithms are motivated by practical applications. A strong field in the area of randomized heuristics is concerned with experimental design (Bartz-Beielstein 2006). Here, the focus lies on deriving guidelines for designing experiments, mainly with the goal of finding better parametrization of the considered algorithms. This is clearly different from our approach since we use experiments to point out the practical relevance of parameter settings derived theoretically.

He et al. (2007) consider measures for the hardness of objective functions and show that predictive measure with polynomial run time do not exist unless P=NP or BPP=NP, respectively. Jansen and Wiegand (2004b) study a previously published empirical research and point out the importance of using existing theoretical results to guide experimental design. Additionally they state that in experimental studies it is highly important to properly state the used parametrization of the algorithm in order to make the results useful in practice. This refers in particular to statements about the expected behavior for increasing problem sizes.

In classical algorithms, the field of algorithm engineering (Sanders 2009) has sparked a lot of attention. It addresses the problem that often the theoretically best algorithms are actually not practical. One reason for this problem are large constants hidden in the asymptotic run times. While in theory often only the order of growth is of interest, constants matter when actually executing an algorithm. Moreover, it is in most cases not sufficient to distinguish between polynomial and exponential run times since the concrete degree of the polynomial matter. Beyond that many algorithms are developed assuming ideal circumstances that are not fulfilled in practice. Sometimes it is, due to complicated incorporated mechanisms or incomplete descriptions, not even reasonable to develop a suitable implementation. Algorithm engineering aims at bridging the gap between theory and practice by developing practical relevant algorithms. Clearly, this includes the implementation of the algorithms and performing experiments. This way, further insights are gained and successive, practical improvements of the algorithms can be obtained. We remark that our approach is different since our main purpose is not yet the development of more practical algorithms to this degree. We are before this important step and contribute to the discussion of theoretical approaches in the theory of randomized search heuristics to make them more practical.

11.2. Contribution of this Thesis

In order to obtain algorithms that are better in practice, it is important to use an appropriate cost model that reflects the 'real' run time of the algorithm. Consider for example sorting algorithms (Cormen et al. 2001). When only counting the number of comparisons executed, we get $O(n \log n)$ as an upper bound for insertion sort. However, in the worst case $\Theta(n^2)$ write operations have to be executed and thus, only taking comparisons into account is not sufficient from a practical point of view.

In this thesis, we revisit the cost models used for the analysis of the optimization time of a randomized search heuristic. Recall, that we simply used the number of iterations the algorithm made until finding a global optimum. In many cases, this corresponds to the number of function evaluations (except for the cost of initialization μ) which is widely accepted as an appropriate measure for the optimization time of a randomized search heuristic. However, when using aging this kind of model seems to be rather coarse since the number of function evaluations per iteration is a random variable itself. Even counting the number of function evaluations becomes doubtful when considering more advanced algorithms, incorporating complicated and expensive mechanisms.

We consider the limitations of both cost models in the following. Since we are aware of the fact that counting iterations might be misleading when considering aging operators, we investigate the gap between these two models using an example discussed in Chapter 10. We show that while we have to be careful when simplifying our cost model, it does not make a huge difference within the analysis executed in this thesis. Afterwards, we demonstrate that even the refined model of counting function evaluations already has limitations when considering precise analyses (in contrast to asymptotic analyses) on simple example functions. We propose a more advanced cost model to address this problem.

12. On Limitations of Counting Function Evaluations

In the preceding chapters of this thesis we have seen several examples for complexity analyses of artificial immune systems. These analyses have in common that the (expected) optimization time is measured by means of the number of iterations or function evaluations needed until a global optimum is found for the first time. These choices are common cost models for the optimization time of randomized search heuristics. Remember, that the reason for counting function evaluations is that randomized search heuristics tend to be algorithmically simple and each step can be carried out relatively quickly. Thus, it is assumed that a function evaluation is the most costly operation and that all other parts of the algorithm are negligible with respect to computation time. Clearly, when only counting the number of iterations, one has to be more careful, since there might be more than one function evaluation per iteration. We discuss this in Section 12.1 using the analyses of aging operators in Part III of this thesis as an example.

The results are usually given in asymptotic form, e. g., $\mathrm{E}(T) = O(f(n))$, $\mathrm{E}(T) = \Omega(f(n))$, or $\mathrm{E}(T) = \Theta(f(n))$ for some function $f\colon \mathbb{N} \to \mathbb{R}_0^+$, where n denotes the length of the bit strings encoding the search points (compare Definition B.2). Remember, that all our results are solely asymptotic in n (not in any other parameter), i. e., they hold for sufficiently large finite values of n. Clearly, asymptotic analysis can distinguish different degrees of efficiency and describe scaling behaviors of the considered algorithms. However, such analyses are not capable of making concrete predictions since the results are quite coarse in nature. Even for an asymptotically tight bound $\mathrm{E}(T) = \Theta(f(n))$ we only know that there exist some constants $n_0 \in \mathbb{N}$ and $c_1, c_2 \in \mathbb{R}^+$ with $0 < c_1 \leq c_2$ such that $c_1 f(n) \leq \mathrm{E}(T) \leq c_2 f(n)$ holds for all $n \geq n_0$. Although the concrete constants n_0, c_1 and c_2 can often be derived from the proofs, predictions about the optimization time (in terms of functions evaluations) are vague (depending on the gap $c_2 - c_1$). A solution for this problem may be the development of more precise results.

Remember that the motivation for the theory of randomized search heuristics is to come up with better heuristics for practical applications. Clearly, practitioners are mainly interested in wall clock time. For this purpose asymptotic results may be too imprecise or even misleading since from this point of view other parts of the randomized search heuristics, e. g., diversity preserving mechanisms or advanced variation operators, might not be negligible as assumed by the considered cost model. Although, such rough results can yield valuable insights (see for example Chapter 9 or Jansen and Sudholt (2010)), they sometimes lead to incorrect conclusions from a practical point of view. Moreover, when trying to overcome the limitations of asymptotic analysis by developing more precise results, these results may even turn out to be misleading instead of helpful.

In the following, we further discuss two aspects of current complexity analyses in order to point out limitations of the cost models used for measuring the optimization time of a randomized search heuristic, namely the number of iterations (Section 12.1) and the number of function evaluations (Section 12.2). Moreover, we present a novel approach introduced in Jansen and Zarges (2011d) that —in the spirit of algorithm engineering (Sanders 2009)— adds empirical analysis to the analytical one in order to bridge the gap between theory and practice in the field of theory of randomized search heuristics (Section 12.3). Sections 12.2 and 12.3 of this chapter are based on Jansen and Zarges (2011d).

12.1. Case Study 1: Counting Iterations

Throughout this thesis we used the number of iterations as a measure for the optimization time of an algorithm. When discussing different mutation operators in Part II this was (except for the cost of initialization) not different from counting the number of function evaluations since in each iteration $\lambda = 1$ new search point is created. Remember, that in this thesis the optimization time is always larger than μ, and thus neglecting these cost is not relevant with respect to asymptotic results. However, when considering aging operators in Part III things change considerably since now there may be more than 1 function evaluation per iteration since newly generated search points introduced due to aging also require function evaluations. As each search point can be removed at most each $\tau_{\max}$-th iteration the number of function evaluations is bounded above by $\mu + (1 + (\mu/\tau_{\max})) \cdot T$ where T is the number of iterations.

We revisit one particular experiment that was executed in Chapter 10 in order to investigate these effects. Remember that in that chapter we analyzed what aging can achieve beyond restarts and thus, the considerations relied heavily on the existence of partial restarts. Clearly, partial restarts increase the number of function evaluations. Note that in the other analyses of aging, we often set the maximal lifespan to values that yield a high probability of not removing any search points due to age in most parts of the optimization process. However, if aging is to be effective, we need to remove search points due to age and thus, it makes sense to consider an example including restarts and partial restarts.

We consider the most frequent replacement strategy (Definition 10.2) that was analyzed in Section 10.5. Moreover, we use the age-based static pure aging variant (Definition 10.1). We choose $\mu = \lfloor\sqrt{n}\rfloor$ as size for the collection of search points and re-do the experiment executed on the objective function f (Definition 10.1.2). We perform 100 independent runs for $n \in \{10, 20, \dots, 200\}$ and count for each run the number of iterations as well as the number of function evaluations executed. The results are depicted in Figure 12.1 using box-and-whisker plots (Definition B.1).

We observe that no difference is visible. Thus, we additionally consider the difference of the observed medians for the number of iterations and the number of function evaluations. The results are presented in Figure 12.2. First, consider the red curve, denoting the exact difference between the two cost measures.

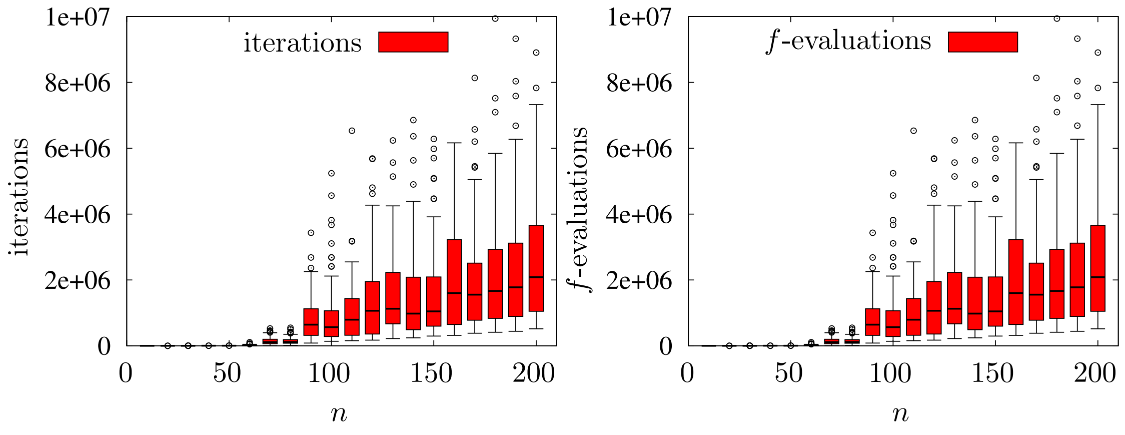


Figure 12.1.: Experimental results for most frequent replacement and age-based aging with $\mu = \lfloor\sqrt{n}\rfloor$, counting iterations (left) and function evaluations (right), data from 100 runs.

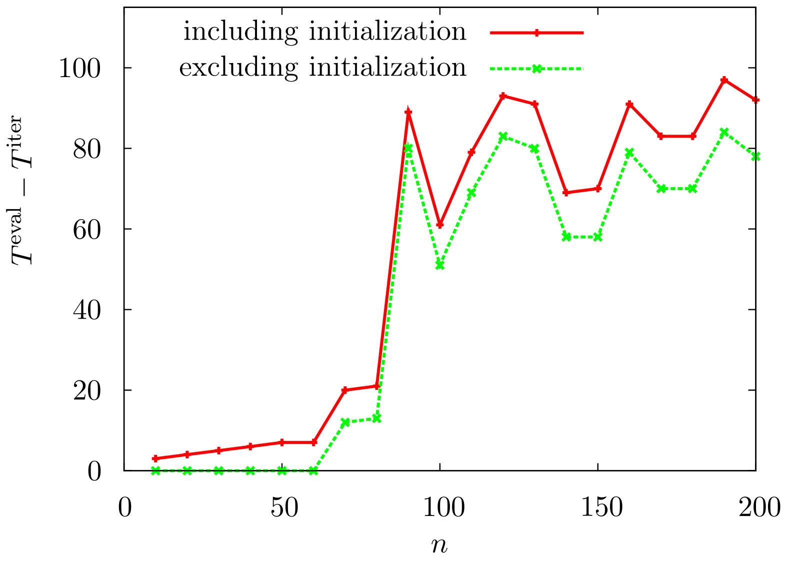


Figure 12.2.: Difference of the observed medians in 100 independent runs when counting iterations (T^{iter}) and function evaluations (T^{eval}), respectively. Differences including and excluding the fixed cost of μ for the initialization are depicted.

We see that we have indeed extra function evaluations. Moreover, their number seems to increase with increasing n, though the difference seems not to bee too large in comparison to the overall optimization time observed. However, we already knew that the number of function evaluations is larger due to the cost of initialization. Thus, we additionally plot the difference of the two cost measures minus μ (green curve). We see that we still have an increasing number of function evaluations compared to the number of iterations.

We conclude that in our considerations counting the number of iterations does not make a decisive difference and thus, the cost model is appropriate for our analyses. However, it is extremely important to keep this discrepancy in mind, when interpreting the results, in particular when comparing age-based algorithms with other algorithms that do not include extra function evaluations per iteration. Moreover, things may change when considering other objective functions or algorithms.

12.2. Case Study 2: Precise Analysis

For our second case study we consider a very recent exact and precise analysis executed by Böttcher et al. (2010). In this analysis, the optimal mutation probability of a well-known evolutionary algorithm, the (1+1) EA (Definition 3.1) on the example function LeadingOnes (Definition 5.8) is analyzed. Note, that the authors consider a general version of the (1+1) EA with a fixed mutation probability $p(n)$ that may (and should) depend on n but is fixed during a run.

It is known that the expected optimization time of the (1+1) EA on LeadingOnes is $\mathrm{E}(T) = \Theta(n^2)$ (Droste et al. 2002) for any mutation probability $p(n) = \Theta(1/n)$. However, for the (1+1) EA on LeadingOnes much more is known, in particular the complete probability distribution for the current bit string x. Let $x \in \{0,1\}^n$ be some search point with $\text{LeadingOnes}(x) = v_x$. Clearly, we have $x[0] = x[1] = \cdots = x[v_x - 1] = 1$ and $x[v_x] = 0$ in this case. If $v_x \leq n-2$, a simple inductive proof (Droste et al. 2002) shows that for any $x' \in \{0,1\}^n$ with $x'[0] = x'[1] = \cdots = x'[v_x - 1] = 1$ and $x'[v_x] = 0$

$$\mathrm{Prob}\left(x = x'\right) = 2^{-(n-(v_x+1))}$$

holds. This can be exploited to derive a much more precise result which was presented recently by Böttcher et al. (2010). For the sake of completeness, we restate their result and the main proof ideas here.

Theorem 12.1 (Böttcher et al. (2010)). *The expected optimization time of the* (1+1) EA *with fixed mutation probability* $p(n)$ *for* LeadingOnes *is*

$$E(T) = \frac{(1-p(n))^{1-n} - (1-p(n))}{2p(n)^2} = \frac{1-(1-p(n))^n}{2p(n)^2(1-p(n))^{n-1}}. \tag{12.1}$$

Proof. Let x be the current search point and $v_x = \text{LeadingOnes}(x)$. Moreover, let T_{v_x} denote the time until an offspring y with $\text{LeadingOnes}(y) > v_x$ is generated for the first time. In order to increase this value in a mutation, it is necessary not to flip the first v_x bits and to flip $x[v_x]$. The remaining bits are irrelevant. Thus, we have $\text{Prob}\left(\textsc{LeadingOnes}(y) > \textsc{LeadingOnes}(x)\right) = (1-p(n))^{v_x} p(n)$ for the mutated search point y. This yields

$$\mathrm{E}\left(T_{v_x}\right) = \frac{1}{(1-p(n))^{v_x} p(n)}. \tag{12.2}$$

Assume x_0 to be the initial search point. Taking random initialization into account and using the law of total probability (Lemma B.11), we can easily express the overall expected optimization time $\mathrm{E}(T)$ via the improvement times T_{v_x}. Note, that the equation in $(*)$ was proven in Böttcher et al. (2010) by induction with a slightly different notation.

$$\begin{aligned}
\mathrm{E}(T) &= \sum_{v_x=0}^{n-1} \text{Prob}\left(\textsc{LeadingOnes}(x_0) = v_x\right) \cdot \mathrm{E}\left(T \mid \textsc{LeadingOnes}(x_0) = v_x\right) \\
&= \sum_{v_x=0}^{n-1} 2^{-(v_x+1)} \left(\mathrm{E}\left(T_{v_x}\right) + \sum_{j=v_x+1}^{n-1} 2^{j-v_x} \cdot \mathrm{E}\left(T \mid \textsc{LeadingOnes}(y) = j\right) \right) \\
&\overset{(*)}{=} \sum_{v_x=0}^{n-1} 2^{-(v_x+1)} \left(\mathrm{E}\left(T_{v_x}\right) + \frac{1}{2} \sum_{j=v_x+1}^{n-1} \mathrm{E}\left(T_j\right) \right) = \sum_{v_x=0}^{n-1} \frac{\mathrm{E}\left(T_{v_x}\right)}{2}
\end{aligned} \tag{12.3}$$

Plugging in eq. (12.2) and making use of Lemma B.7 on the geometric series we conclude the proof.

$$\begin{aligned}
\mathrm{E}(T) &= \sum_{v_x=0}^{n-1} \frac{1}{2(1-p(n))^{v_x} p(n)} = \frac{1}{2p(n)} \sum_{v_x=0}^{n-1} \left(\frac{1}{1-p(n)} \right)^{v_x} \\
&= \frac{1}{2p(n)} \cdot \frac{1 - \left(\frac{1}{1-p(n)}\right)^n}{1 - \frac{1}{1-p(n)}} = \frac{1}{2p(n)} \cdot \frac{(1-p(n))^n - 1}{(1-p(n))^n} \cdot \frac{1-p(n)}{1-p(n)-1} \\
&= \frac{1-(1-p(n))^n}{2p(n)^2(1-p(n))^{n-1}}
\end{aligned}$$

□

Using this result Böttcher et al. (2010) derive an optimal fixed mutation probability for the (1+1) EA on LeadingOnes. It is stated that the mutation probability minimizing eq. (12.1) converges to $1.59362/n$, yielding $\mathrm{E}(T) = 0.772n^2$. It is important to notice that the optimal mutation probability depends on n and only converges to $1.59362/n$ for increasing n. To be more precise, for small values of n slightly smaller mutation probabilities lead to better optimization times, e. g., $1.58105/n$ for $n = 100$ and $1.59235/n$ for $n = 1000$. However, for not too small values of n all significantly other mutation probabilities yield larger expected optimization times. Note, that when doing experiments, we use the exact optimal values that can be derived using eq (12.1). These values are visualized later in Figure 12.6 (together with much smaller values from the new cost model introduced later in Section 12.3).

Algorithm 12.1 Implementation of LeadingOnes (Jansen and Zarges 2011d)

```
long fitness(char *x)
{
   long i;
   for ( i=0; ( i<n ) && ( x[i]==1 ); i++ );
   return i;
}
```

Remember that the cost model used in the presented analysis only counts the number of function evaluations since it assumes those to be most costly. However, at the end of the day time is wall clock time. In particular, practitioners are mostly interested in the actual run time. Since the main motivation for considering randomized search heuristics is in practical applications, we investigate the result on the optimal mutations probability for LeadingOnes with respect to its relevance when measuring the actual run time instead of the number of function evaluations. In order to do so, it makes sense to first look at the concrete implementation used.

We consider a simple implementation that is neither naive nor sophisticated. In particular, we want an implementation that is organized in modules and can be considered as reasonable from a programming or software engineering point of view. This is useful since in applications randomized search heuristics often exhibit the need to be modified to better fit the current task. One modification is the application of different kinds of crossover and mutation operators or selection schemes. Such modifications are facilitated when the randomized search heuristic is implemented using such a module structure.

The modules we use are function evaluation, mutation, and selection. The function evaluation receives as input a search point and returns its fitness. The mutation receives a search point to be mutated and sufficient memory to store the mutated offspring. In addition to this offspring it returns the information if at least one bit was flipped. No other information (like the specific bits flipped) is available to the main loop. We remark that use of such additional information may enable one to implement a more efficient function evaluation. In our model this is not possible. The selection receives the parent, the offspring, and their fitness values. It returns the new collection of search points together with its fitness.

The implementation is carried out in ANSI C using a `char` to store a single bit. The random decisions of the (1+1) EA are implemented using `drand48()` as pseudo-random number generator. This leads to the straight-forward implementation of the fitness function LeadingOnes as shown in Algorithm 12.1, where the string length n is a global variable.

For the mutation, a naive implementation may perform a random experiment for each of the n bits to determine if this bit is flipped. This would make each mutation an operation that requires $\Theta(n)$ random decisions and thus extremely costly. However, a much more efficient implementation is known. Already Knuth (1969) (described for standard bit mutations by Rudolph and Ziegenhirt (1997)) pointed out that for small

Algorithm 12.2 Implementation of mutation (Jansen and Zarges 2011d)

```
int mutation(char *parent,char *offspring)
{
   long next, start=0; /* start at offspring[0] */
   int mutated=0; / remember if bit is flipped /
   next=getNextPos(n-1); /* get mutation site */
   if ( next != -1 )
   { /* position within current bit string */
      mutated=1; /* remember some bit is mutated */
      /* copy parent to offspring */
      memcpy(offspring, parent, sizeof(char)*n);
      while (next != -1) /* while within string */
      { /* mutate bit */
         offspring[start+next]=1-parent[start+next];
         start += (next+1); /* update next pos.  */
         next=getNextPos(n-start-1);
      }
   }
   return mutated; /* flag if any bit was mutated */
}
```

probabilities $p(n)$ it is much more efficient not to perform n random experiments (one for each potential mutation site) but to randomly draw the position of the next mutation site. This does not change the probability distribution but reduces the number of random experiments necessary from n on expectation to $p(n) \cdot n$, thus only 1 for the mutation probability $p(n) = 1/n$. We implement this idea in a straight-forward way using two global variables, `nextPos` to store the next mutation site and `l=log(1.0-p)` where `p` $= p(n)$ is the mutation probability. This variable `l` is needed for the determination of the next mutation site. We initialize `nextPos=-1` to indicate that currently there is no random position available and one has to be determined randomly. The mutation itself is described in Algorithm 12.2. It makes use of two auxiliary functions described in Algorithm 12.3 and Algorithm 12.4, respectively.

What remains is selection. The strict plus-selection employed requires a comparison of the two function values and, in case the offspring is no worse than its parent, replacing the parent by the offspring. In a naive implementation one may copy the offspring to the parent requiring $\Theta(n)$ computation steps. It is, however, quite obvious that it suffices to swap the pointers to parent and offspring so that selection can be done in time $\Theta(1)$.

All experiments reported in this chapter have been carried out on an iMac with a 2.8GHz quad-core Intel Core i7 processor with 8MB shared L3 cache and 8GB 1066MHz DDR3 SDRAM. The ANSI C implementation has been compiled using gcc 4.2.1 with optimization -O3. The run time is measured using `clock()` so that the actual time spent in the (1+1) EA is measured.

Algorithm 12.3 Implementation of getNextPos (for mutation) (Jansen and Zarges 2011d)

```
long getNextPos(long length)
{ /* deliver next mutation site */
   if (nextPos>=0) /* next position available */
      return savePos(nextPos, length);
   /* randomly choose next position */
   nextPos=(long)floor( log( drand48() )/l );
   return savePos(nextPos, length);
}
```

Algorithm 12.4 Implementation of savePos (for mutation) (Jansen and Zarges 2011d)

```
long savePos(long pos, long length)
{ /* update position */
   if (pos>length)
   { /* next position not within string */
      nextPos = pos-length-1; /* subtract used part */
      return -1; /* signal:  end for this string */
   }
   else
   {
      nextPos=-1; /* position used */
      return pos;
   }
}
```

The experiments are all carried out for $n \in \{50, 100, 150, \dots, 1000\}$ and are repeated 100 times independently, i. e., with independent random seeds for the pseudo-random number generator. The results are always presented using box-and-whisker plots as described in the appendix (Definition B.1).

Using the implementation and experimental setup described above, we compare the number of function evaluations as well as the actual run time of the (1+1) EA with the standard mutation probability $p(n) = 1/n$ and the optimal mutation probability derived by Böttcher et al. (2010). Note, that the expected optimization times with respect to function evaluations are $\Theta(n^2)$ in both cases and thus, only differ in the leading constant.

The results of our experiments can be found in Figure 12.3. We see that with respect to the number of function evaluations, the results are in accordance to the theoretical results (Figure 12.3(a)). We additionally plot the graph of the function cn^2, where the constant c is determined by a least squares fit for the means and get a reasonable fit for $0.77n^2$ and $0.86n^2$, respectively. Note, that this matches the expected optimization times due to Theorem 12.1 for the considered mutation probabilities.

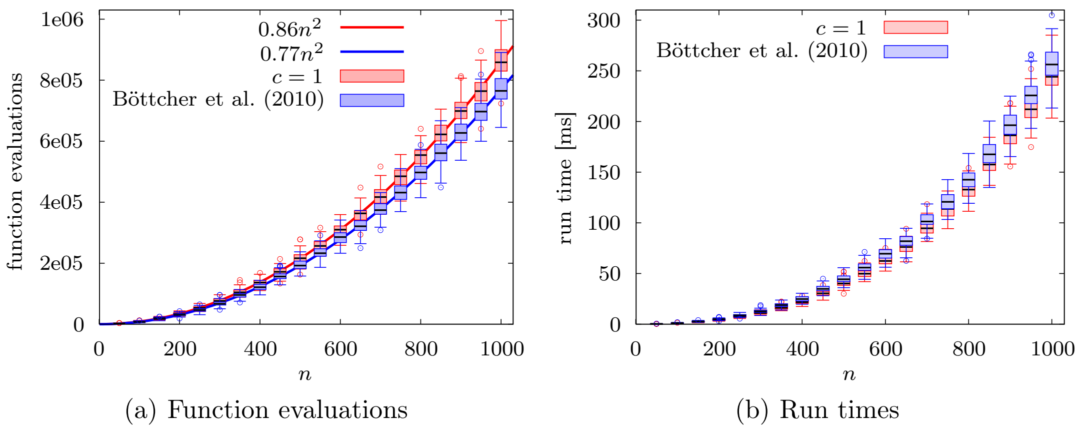


Figure 12.3.: Comparison of the (1+1) EA for LeadingOnes using the standard mutation probability $p(n) = 1/n$ and the optimal mutation probability due to Böttcher et al. (2010), data from 100 runs.

However, things change, when considering the actual run time of the algorithm (see Figure 12.3(b)). Contrary to expectations, the theoretical optimal mutation probability does not seem to outperform the standard choice $p(n) = 1/n$ in the considered setting, it even seems to perform slightly worse. Note, that we have to be careful with our conclusions here due to several reasons. First, we are considering a specific implementation of the algorithm and results may be different for other implementations. Second, the observed run times in the experiments are all very small in nature. Third, the differences are likely to be not statistically significant since they are too small (in particular the box-and-whisker plots are overlapping). However, we can conclude that, when performing precise analyses in the way described above, theoretical findings may be misleading with respect to the actual run time of the considered algorithm. We conjecture that the number of function evaluations is a too coarse and inaccurate measure of the run time and thus, only counting the number of function evaluations is not an appropriate cost model for such analyses. Since implementation details can make a huge difference, we propose a different cost model within the next section that takes these details into account in order to improve the practical relevance of the theoretical results.

12.3. A Data-Driven Approach

In this section, we introduce a novel approach for measuring the optimization time of a randomized search heuristics that incorporates empirical findings into the theoretical analysis (Jansen and Zarges 2011d). It is important to note, that we do not argue against state-of-the-art computational complexity analysis in general. We rather make the point that an analysis that takes the concrete implementation of the algorithm into account can yield insights that a pure theoretical optimization time analysis cannot.

First remember that the standard cost model for the analysis of randomized search heuristics counts only the number of function evaluations since it assumes those to be most costly. Taking a closer look at the implementation described in the preceding section, the first observation that was made in Jansen and Zarges (2011d), is that a function evaluation here is not at all time consuming since it can be carried out in $O(n)$ steps and actually takes only $\Theta(\textsc{LeadingOnes}(x))$ steps. Since with high probability in each mutation the number of leading 1-bits is increased by $O(1)$, we have that on average

$$\Theta\left(\left(\sum_{i=1}^{n} i\right)/n\right) = \Theta(n)$$

computation steps are carried out in a function evaluation where the average is taken over the complete run. Moreover, it is noted, that being able to carry out one function evaluation in time $O(n)$ is not unusual. Most example functions can be computed in linear time (even more complex ones like the well-known long path function (Rudolph 1996, 1997), compare also Chapter 8.2) and the same holds for many combinatorial optimization problems. It is true that function evaluations can be time consuming. However, in many cases and in particular for LeadingOnes, they are not. Considering the rest of the (1+1) EA mutation seems to be a potentially time consuming operator since it involves pseudo-random number generation and the computation of a logarithm.

However, in a first step we work under the hypothesis that the assumption that the function evaluations account for the majority of the actual run time is correct. In this case, any sensible implementation will be careful with respect to function evaluations. Consider the (1+1) EA with the standard choice $p(n) = 1/n$. We see that with probability $(1 - p(n))^n > .35$ for $n > 10$ no bit is flipped at all and $y = x$ holds. Thus, in expectation, in more than 35% of the iterations no bit is flipped. Clearly, in these cases no function evaluation is necessary. Note, that the implementation described in Section 12.2 allows to omit a function evaluation when producing a copy, since the mutation returns the additional information if at least one bit was flipped. We remark that this kind of resampling is typical for randomized search heuristics in discrete search spaces and completely different from randomized search heuristics in continuous domains. The proposed approach takes this resampling into account.

12.3.1. Towards an Advanced Cost Model

We propose an advanced cost model reflecting the above observation by assigning cost $q(n)$ to an iteration where in the mutation no bit is flipped and cost $r(n)$ to an iteration where at least one bit is flipped. Using this cost model we revisit the proof of Theorem 12.1 and derive the following more general result. Note, that setting $q(n) := r(n) := 1$ yields the cost model used in Theorem 12.1 by Böttcher et al. (2010).

Theorem 12.2 (Jansen and Zarges (2011d)). *The expected optimization time of the* (1+1) EA *with fixed mutation probability* $p(n)$ *for* LeadingOnes *is*

$$E(T) = (1 - (1 - p(n))^n) \cdot \frac{(1 - p(n))^n q(n) + (1 - (1 - p(n))^n)\, r(n)}{2p(n)^2(1 - p(n))^{n-1}}. \qquad (12.4)$$

Proof. Let x be the current bit string of the (1+1) EA and $v_x = \textsc{LeadingOnes}(x) < n$. Let G_{v_x} denote the number of iterations until an offspring y with $\textsc{LeadingOnes}(y) > v_x$ is generated for the first time. Let T_{v_x} denote the costs in these iterations. For $i \in \mathbb{N}$, let $T_{v_x,i}$ denote the costs in the i-th iteration. Note, that here, T_{v_x} and $T_{v_x,i}$ correspond to the costs (instead of the number of function evaluations) with respect to the cost model and, thus, $T_{v_x,i} \in \{q(n), r(n)\}$.

Applying the law of total probability (Lemma B.11) we have

$$\mathrm{E}\left(T_{v_x}\right) = \sum_{t=1}^{\infty} \mathrm{Prob}\left(G_{v_x} = t\right) \cdot \mathrm{E}\left(T_{v_x} \mid G_{v_x} = t\right).$$

Moreover, by definition $T_{v_x} = \sum\limits_{i=1}^{G_{v_x}} T_{v_x,i}$ holds and thus,

$$\mathrm{E}\left(T_{v_x}\right) = \sum_{t=1}^{\infty} \mathrm{Prob}\left(G_{v_x} = t\right) \cdot \sum_{i=1}^{t} \mathrm{E}\left(T_{v_x,i} \mid G_{v_x} = t\right)$$

follows by linearity of expectation (Lemma B.12). For $i = G_{v_x}$ we have $\mathrm{E}\left(T_{v_x,i}\right) = r(n)$ since in this final mutation at least the left-most 0-bit flips. For $i < G_{v_x}$ all $\mathrm{E}\left(T_{v_x,i}\right)$ are equal for symmetry reasons and thus

$$\mathrm{E}\left(T_{v_x}\right) = \sum_{t=1}^{\infty} \mathrm{Prob}\left(G_{v_x} = t\right) \cdot \left((t-1)\mathrm{E}\left(T_{v_x,1} \mid G_{v_x} = t\right) + r(n)\right) \tag{12.5}$$

holds. We have $G_{v_x} = t$ if and only if on the one hand in the t-th iteration the left-most v_x bits do not flip and $x[v_x]$ flips and, on the other hand, this does no happen in the $t-1$ preceding iterations. Thus

$$\mathrm{Prob}\left(G_{v_x} = t\right) = \left(1 - p(n)(1-p(n))^{v_x}\right)^{t-1} \cdot p(n)(1-p(n))^{v_x} \tag{12.6}$$

holds. For $i < G_{v_x}$ we consider one single mutation. Let S denote the event that in this mutation the function value is increased. Let Z denote the event that in this mutation no bit is flipped. Since we consider $T_{v_x,i}$ for $i < G_{v_x}$ we know that the function value is not increased. For $\mathrm{Prob}\left(\overline{Z} \wedge \overline{S}\right)$ we observe $\mathrm{Prob}\left(\overline{S}\right) = \mathrm{Prob}\left(\overline{Z} \wedge \overline{S}\right) + \mathrm{Prob}\left(Z\right)$ and $\mathrm{Prob}\left(\overline{Z} \wedge \overline{S}\right) = \mathrm{Prob}\left(\overline{S}\right) - \mathrm{Prob}\left(Z\right)$ follows. Thus, using Definition B.3,

$$\begin{aligned}
\mathrm{E}\left(T_{v_x,i}\right) &= q(n) \cdot \mathrm{Prob}\left(Z \mid \overline{S}\right) + r(n) \cdot \mathrm{Prob}\left(\overline{Z} \mid \overline{S}\right) \\
&= q(n) \cdot \frac{\mathrm{Prob}\left(Z \wedge \overline{S}\right)}{\mathrm{Prob}\left(\overline{S}\right)} + r(n) \cdot \frac{\mathrm{Prob}\left(\overline{Z} \wedge \overline{S}\right)}{\mathrm{Prob}\left(\overline{S}\right)} \\
&= q(n) \cdot \frac{\mathrm{Prob}\left(Z\right)}{\mathrm{Prob}\left(\overline{S}\right)} + r(n) \cdot \frac{\mathrm{Prob}\left(\overline{S}\right) - \mathrm{Prob}\left(Z\right)}{\mathrm{Prob}\left(\overline{S}\right)}
\end{aligned}$$

for all $i < G_{v_x}$ holds. We already know that $\text{Prob}\left(\overline{S}\right) = 1 - \text{Prob}\left(S\right) = 1 - (1-p(n))^{v_x} \cdot p(n)$ holds. With $\text{Prob}\left(Z\right) = (1-p(n))^n$ we obtain

$$\text{E}\left(T_{v_x,i}\right) = q(n) \cdot \frac{(1-p(n))^n}{1-(1-p(n))^{v_x} \cdot p(n)} + r(n) \cdot \left(1 - \frac{(1-p(n))^n}{1-(1-p(n))^{v_x} \cdot p(n)}\right).$$

Inserting this value for $\text{E}\left(T_{v_x,i}\right)$ and $\text{Prob}\left(G_{v_x} = t\right)$ from eq. (12.6) into eq. (12.5) we obtain

$$\begin{aligned}
&\text{E}\left(T_{v_x}\right) \\
&= \sum_{t=1}^{\infty} \left(1 - p(n)(1-p(n))^{v_x}\right)^{t-1} p(n)(1-p(n))^{v_x} \\
&\qquad \cdot \left((t-1)\left(\frac{q(n)(1-p(n))^n}{1-p(n)(1-p(n))^{v_x}} + r(n)\left(1 - \frac{(1-p(n))^n}{1-p(n)(1-p(n))^{v_x}}\right)\right) + r(n)\right) \\
&= \sum_{t=1}^{\infty} \left(1 - p(n)(1-p(n))^{v_x}\right)^{t-1} p(n)(1-p(n))^{v_x} \\
&\qquad \cdot \left((t-1)\left(\frac{q(n)(1-p(n))^n}{1-p(n)(1-p(n))^{v_x}} + r(n)\left(1 - \frac{(1-p(n))^n}{1-p(n)(1-p(n))^{v_x}}\right)\right)\right) \\
&\quad + \sum_{t=1}^{\infty} \left(1 - p(n)(1-p(n))^{v_x}\right)^{t-1} p(n)(1-p(n))^{v_x} r(n) \\
&= p(n)(1-p(n))^{v_x} \left(\frac{q(n)(1-p(n))^n}{1-p(n)(1-p(n))^{v_x}} + r(n)\left(1 - \frac{(1-p(n))^n}{1-p(n)(1-p(n))^{v_x}}\right)\right) \\
&\qquad \cdot \sum_{t=1}^{\infty} (t-1)\left(1 - p(n)(1-p(n))^{v_x}\right)^{t-1} \\
&\quad + p(n)(1-p(n))^{v_x} r(n) \sum_{t=1}^{\infty} \left(1 - p(n)(1-p(n))^{v_x}\right)^{t-1}
\end{aligned}$$

It is easy to see that the two series in the above equations converge since we can apply Lemma (B.7) on the geometric series. This yields

$$\begin{aligned}
&\sum_{t=1}^{\infty} (t-1)\left(1 - p(n)(1-p(n))^{v_x}\right)^{t-1} = \sum_{t=1}^{\infty} t\left(1 - p(n)(1-p(n))^{v_x}\right)^{t} \\
&= \sum_{t=1}^{\infty} \sum_{u=t}^{\infty} \left(1 - p(n)(1-p(n))^{v_x}\right)^{u} = \sum_{t=1}^{\infty} \frac{\left(1 - p(n)(1-p(n))^{v_x}\right)^{t}}{1 - \left(1 - p(n)(1-p(n))^{v_x}\right)} \\
&= \frac{1}{p(n)(1-p(n))^{v_x}} \sum_{t=1}^{\infty} \left(1 - p(n)(1-p(n))^{v_x}\right)^{t} \\
&= \frac{1}{p(n)(1-p(n))^{v_x}} \cdot \frac{1 - p(n)(1-p(n))^{v_x}}{1 - \left(1 - p(n)(1-p(n))^{v_x}\right)} = \frac{1 - p(n)(1-p(n))^{v_x}}{\left(p(n)(1-p(n))^{v_x}\right)^2}
\end{aligned}$$

for the first term and

$$\begin{aligned}\sum_{t=1}^{\infty}(1-p(n)(1-p(n))^{v_x})^{t-1} &= \sum_{t=0}^{\infty}(1-p(n)(1-p(n))^{v_x})^{t}\\ = \frac{1}{1-(1-p(n)(1-p(n))^{v_x})} &= \frac{1}{p(n)(1-p(n))^{v_x}}\end{aligned}$$

for the second one. Plugging these results into the above calculations we obtain

$$\begin{aligned}&\mathrm{E}\left(T_{v_x}\right)\\ &= p(n)(1-p(n))^{v_x}\left(\frac{q(n)(1-p(n))^n}{1-p(n)(1-p(n))^{v_x}} + r(n)\left(1-\frac{(1-p(n))^n}{1-p(n)(1-p(n))^{v_x}}\right)\right)\\ &\quad \cdot \frac{1-p(n)(1-p(n))^{v_x}}{(p(n)(1-p(n))^{v_x})^2} + \frac{p(n)(1-p(n))^{v_x} r(n)}{p(n)(1-p(n))^{v_x}}\\ &= \frac{1-p(n)(1-p(n))^{v_x}}{p(n)(1-p(n))^{v_x}}\left(\frac{q(n)(1-p(n))^n}{1-p(n)(1-p(n))^{v_x}} + r(n)\left(1-\frac{(1-p(n))^n}{1-p(n)(1-p(n))^{v_x}}\right)\right)\\ &\quad + r(n)\\ &= \frac{1-p(n)(1-p(n))^{v_x}}{p(n)(1-p(n))^{v_x}} \cdot \frac{q(n)(1-p(n))^n}{1-p(n)(1-p(n))^{v_x}}\\ &\quad + \frac{r(n)\left(1-p(n)(1-p(n))^{v_x}\right)}{p(n)(1-p(n))^{v_x}} \cdot \left(1-\frac{(1-p(n))^n}{1-p(n)(1-p(n))^{v_x}}\right) + r(n)\\ &= \frac{q(n)(1-p(n))^n}{p(n)(1-p(n))^{v_x}} + \frac{r(n)\left(1-p(n)(1-p(n))^{v_x}\right)}{p(n)(1-p(n))^{v_x}} - \frac{r(n)(1-p(n))^n}{p(n)(1-p(n))^{v_x}} + r(n)\\ &= \frac{(1-p(n))^n q(n) + \left(1-(1-p(n))^n\right) r(n)}{p(n)(1-p(n))^{v_x}}\end{aligned}$$

Making use of eq. (12.3) from the proof of Theorem 12.1 we obtain

$$\begin{aligned}\mathrm{E}\left(T\right) &= \sum_{v_x=0}^{n-1}\frac{(1-p(n))^n q(n) + \left(1-(1-p(n))^n\right) r(n)}{2p(n)(1-p(n))^{v_x}}\\ &= \frac{(1-p(n))^n q(n) + \left(1-(1-p(n))^n\right) r(n)}{2p(n)} \cdot \sum_{v_x=0}^{n-1}\frac{1}{(1-p(n))^{v_x}}\\ &= \frac{(1-p(n))^n q(n) + \left(1-(1-p(n))^n\right) r(n)}{2p(n)} \cdot \frac{1-(1-p(n))^n}{p(n)(1-p(n))^{n-1}}\\ &= \left(1-(1-p(n))^n\right)\frac{(1-p(n))^n q(n) + \left(1-(1-p(n))^n\right) r(n)}{2p(n)^2(1-p(n))^{n-1}}\end{aligned}$$

for the expected total cost T. □

We still need to discuss how to determine $q(n)$ and $r(n)$ appropriately. Therefore, we consider two different instantiations of the cost model within the next two sections.

12.3.2. Naive Analysis

In a first step, we examine a very simple version. We keep the cost model where we assign cost 1 for a function evaluation but take into account that no function evaluation is made if no bit is flipped. This corresponds to the case $q(n) = 0$ and $r(n) = 1$. Again, let T denote the cost in this modified cost model. The following holds.

Theorem 12.3 (Jansen and Zarges (2011d))**.** *Let* $p(n)$ *with* $0 < p(n) < 1$ *denote the mutation probability of the* (1+1) EA*,* $c > 0$ *a constant.*

$$p(n) = c/n \Rightarrow \lim_{n\to\infty} E(T)/n^2 = \frac{(e^c-1)^2}{2c^2e^c} > .5 \tag{12.7}$$

$$p(n) \geq \frac{1}{n} \Rightarrow \lim_{n\to\infty} E(T)/n^2 \geq \frac{(e-1)^2}{2e} > .54 \tag{12.8}$$

$$p(n) = o\left(\frac{1}{n}\right) \Rightarrow \lim_{n\to\infty} E(T)/n^2 = \frac{1}{2} = .5 \tag{12.9}$$

Proof. Plugging $q(n) = 0$ and $r(n) = 1$ into eq. (12.2), we get

$$\mathrm{E}(T) = \frac{(1-(1-p(n))^n)^2}{2p(n)^2(1-p(n))^{n-1}}.$$

First we consider the case $p(n) = c/n$ and make use of $\lim_{n\to\infty}(1-c/n)^n = e^{-c}$. Hence, we have

$$\lim_{n\to\infty} \mathrm{E}(T)/n^2 = \frac{(1-e^{-c})^2}{2c^2e^{-c}} = \frac{(e^c-1)^2}{2c^2e^c}$$

and (12.7) follows. We observe that $\mathrm{E}(T_{v_x})$ grows with growing $p(n) \geq 1/n$ and thus (12.8) holds. For $p(n) = o(1/n)$ we have $p(n) = 1/(n\alpha(n))$ for some function α with $\lim_{n\to\infty} 1/\alpha(n) = 0$. Making use of $\lim_{n\to\infty}(1-p(n))^n = \lim_{n\to\infty} 1-(1/\alpha(n)) = 1$ in this case we obtain (12.9). $\square$

We visualize the limit terms (as a function of c) from Theorem 12.3 together with the constant term 0.772 that results from using the fixed mutation probability $1.59362/n$ (Böttcher et al. 2010) in Figure 12.4. Theorem 12.3 tells us that the number of function evaluations can be minimized by making the mutation probability arbitrarily small, e. g., using $p(n) = 2^{-n}$ implies a smaller number of function evaluations than $p(n) = 1/n$ or $p(n) = 1.59362/n$. This demonstrates that this simple cost model is inadequate since certainly, the mutation probability should not be set this small. We learn that while function evaluations may be costly the other operations of the (1+1) EA are definitely not for free. In order to take them into account in an appropriate way the implementation needs to be taken into account. We do so in the following section.

12.3.3. Data-Driven Cost Model Generation

In theory, it appears to be appropriate to assign cost n to a mutation that flips a bit (due to the function evaluation involved) and cost 1 to a mutation where no bit is

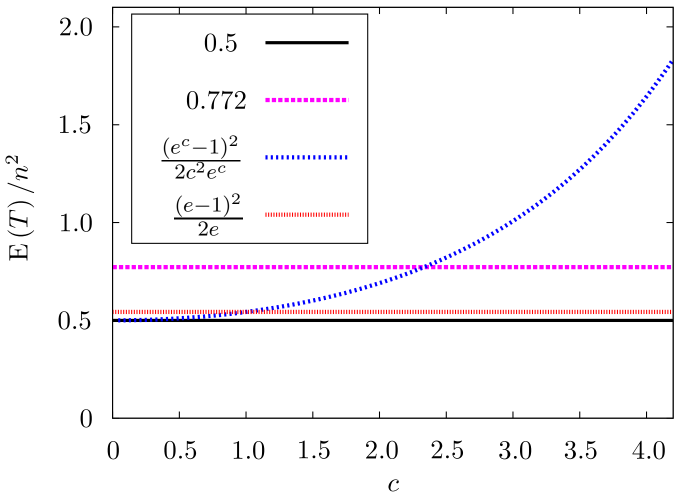


Figure 12.4.: Limit terms for $\lim_{n\to\infty} E(T_{>0})/n^2$ for different values of the mutation probability $p(n)$ as given in Theorem 12.3 together with 0.772 from Böttcher et al. (2010).

flipped. It is, however, unclear if this is a reasonable setting when considering the implemented (1+1) EA since function evaluations involve only simple and fast basic operations whereas mutations involve potentially costly pseudo-random number generation and the computation of a logarithm. Therefore, we use results of preliminary experiments to assign cost to iterations with and without actual mutation as presented in Jansen and Zarges (2011d).

We aim at deriving an estimate of the time spent in iterations without any flipping bit in comparison to iterations where at least one bit flips. We perform 100 independent runs of the (1+1) EA with the standard mutation probability $p(n) = 1/n$ on LeadingOnes. It is important to remark, that the times we are interested in are not independent of the mutation probability $p(n)$. In particular, very small mutation probabilities increase the probability to have several consecutive iterations where no bit is flipped. In all but the first of the iterations of such a sequence the mutation is particularly fast because no random experiment needs to be carried out at all. However, preliminary experiments confirmed that the differences are not significant for mutation probabilities $p(n) = c/n$ with $1/4 < c < 4$. Thus, we do these measurements for $p(n) = 1/n$, only. We discuss larger and smaller mutation probabilities later.

We measure the cumulative time for all iterations where no bit flips and the cumulative time for all iterations where at least one bit flips. For each n we report the quotient of the corresponding average times in form of box-and-whisker plots in Figure 12.5. Since we average over an enormous number of random experiments (100 independent runs where already one run yields the average over a large random number of iterations), the results are concentrated around the mean values in an extreme way. Fitting $ax + b$ by a least squares fit for the means yields a reasonable fit for $0.0011n + 1.18$ (compare

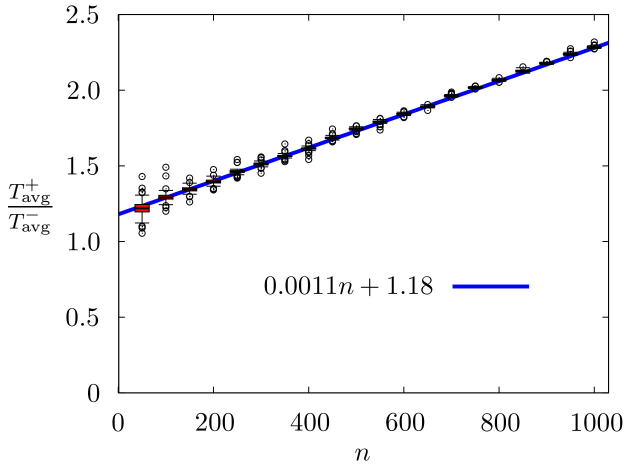


Figure 12.5.: Quotients of measured average run times for iterations with flipping bit (T^+_{avg}) and without flipping bit (T^-_{avg}) , data from 100 runs, using mutation probability $p(n) = 1/n$.

Figure 12.5). We conclude that the relation between the average run time in iterations with and without actual mutation is indeed linear but the factor is really small and hence the ratio grows only slowly with n.

12.3.4. Application of the New Cost Model

Based on these empirical findings we assign cost $q(n) := 1$ to an iteration where in the mutation no bit is flipped and cost $r(n) := 0.0011n + 1.18$ to an iteration where at least one bit is flipped. Plugging these values into eq. (12.4) yields

$$\mathrm{E}\,(T) = (1 - (1 - p(n))^n) \cdot \frac{(1 - p(n))^n q(n) + (1 - (1 - p(n))^n)\,(0.0011n + 1.18)}{2p(n)^2(1 - p(n))^{n-1}} \quad .$$

Similar to Böttcher et al. (2010) we want to derive an optimal mutation probability $p(n)$ with respect to our new cost model, i. e., a mutation probability $p(n)$ that minimizes the expected cost derived in the new cost model. As already stated by Böttcher et al. (2010) we cannot solve the equation for the expected optimization time from Theorem (12.2) arithmetically for $p(n)$. Consequently, we do so numerically for the values of n used in our experiments. We visualize the resulting values $n \cdot p(n)$ together with the corresponding values from Böttcher et al. (2010) and the asymptotic value 1.59362 in Figure 12.6 to allow for comparison.

We see that the optimal mutation probabilities with respect to our empirical cost model decrease with increasing n. Moreover, they are always strictly smaller than the optimal values derived in the uniform cost model applied by Böttcher et al. (2010). We see that for the range of values of n considered the optimal mutation probabilities are all $1/n \leq p(n) \leq 1.5/n$ and thus, the differences resulting from mutation probabilities c/n

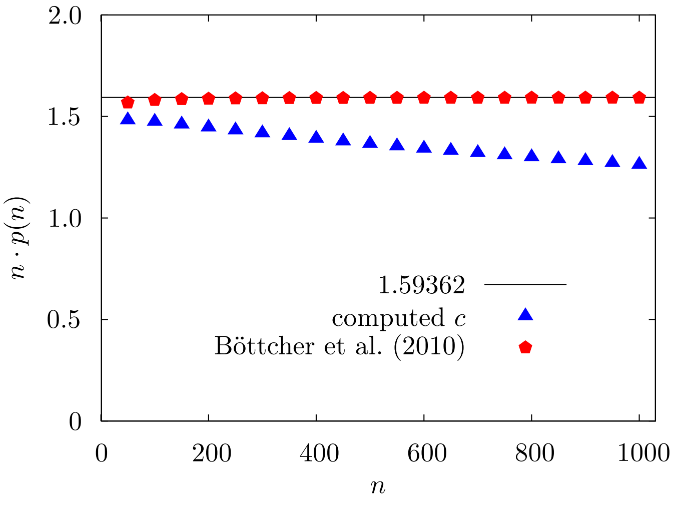


Figure 12.6.: Graph of $n \cdot p(n)$ for mutation probabilities $p(n)$ minimizing the empirical cost model and for mutation probabilities due to Böttcher et al. (2010).

with $1 < c \leq 1.5$ as compared to mutation probability $1/n$ are rather small. This supports our assumption that performing measurements with $p(n) = 1/n$ is indeed sufficient. We come back to this point again when discussing actual run times. It is important to remark that these findings are only valid for the range of values n considered here. For much larger values of n one can expect even smaller mutation probabilities to be optimal.

12.3.5. Experimental Evaluation

To compare the impact of the different mutation probabilities, we perform experiments, again for $n \in \{50, \ldots, 1000\}$ and 100 independent runs for each n and each considered mutation probability.

In addition to the derived optimal mutation probabilities for the empirical cost model (Figure 12.6) and the optimal mutation probability $\approx 1.59362/n$ for the uniform cost model, we consider the most recommended standard choice $1/n$ and several smaller and larger mutation probabilities. Recall that we consider the (1+1) EA on LeadingOnes. The probability to increase the function value equals $(1 - p(n))^{v_x} p(n)$ if v_x denotes the function value of the current bit string. We consider fixed mutation probabilities of the form $c(n)/n$. In most cases $c(n) = c$ is independent of n. For the optimal mutation probabilities, however, $c(n)$ actually depends on n. With mutation probability $p(n) = c(n)/n$ and using Lemma B.8 the expected waiting time for increasing the function value equals

$$\frac{n}{c(n)} \cdot \left(1 - \frac{c(n)}{n}\right)^{-v_x} \geq n \cdot \frac{e^{c(n) \cdot (v_x/n)}}{c(n)}. \tag{12.10}$$

Remember that at some point of time during the run we will have $v_x/n = 1 - O(1/n)$. We see that increasing $c(n)$ beyond 1 increases the expected waiting time exponentially whereas decreasing $c(n)$ below 1 increases it only linearly. Thus, we can expect much more dramatic effects when increasing the mutation probability. Thus, we consider the mutation probabilities c/n and $2^{1-c}/n$ for $c \in \{2, 3, 4, 5\}$ and expect to see a similar increase in run time for larger and smaller mutation probabilities. Remember that we report actual run times and *not* the number of function evaluations. In all plots we have on the x-axis the length of the bit string n and on the y-axis run time in milliseconds.

The results for the different mutation probabilities $p(n) = c/n$ are given as box-and-whisker plots in a number of separate diagrams, all in Figure 12.7. In addition to the choices $c \in \{1/16,\ 1/8,\ 1/4,\ 1/2,\ 1,\ 2,\ 3,\ 4,\ 5\}$, the values due to Böttcher et al. (2010), and the computed values from Figure 12.6 we present results for the mutation probability $p(n) = 1/n^2$ (equivalent to setting $c = 1/n$ in the mutation probability $p(n) = c/n$). This very small mutation probability is motivated in the following way. If we follow the reasoning that an iteration that does not flip any bit can be carried out in time $\Theta(1)$ and other iterations require time $\Theta(n)$ we could assign cost 1 to iterations without flipping bits and cost n to other iterations. Since increasing the function value has always probability $\Theta(1/n)$ we can expect on average to perform $\Theta(n)$ iterations before an improvement occurs. Since in the simple cost model that we consider now these $\Theta(n)$ iterations account for a total cost of $\Theta\big(n^2\big)$ we can afford to have $\Theta\big(n^2\big)$ iterations without mutating any bit without increasing the total cost asymptotically. Thus, from an asymptotic and theoretical point of view, $\Theta\big(1/n^2\big)$ are the smallest mutation probabilities that are still of optimal efficiency on LeadingOnes.

In Figure 12.7 we have one plot for each value of c we consider. In order to allow for some comparison in all plots the same scale is used. We notice that with all mutation probabilities the run times are very much concentrated around the mean value, the box-and-whisker plots are almost collapsed to this value. Moreover, we see that all mutation probabilities $p(n)$ with $1/(16n) \leq p(n) \leq 2/n$ lead roughly to the same run time behavior. This includes the two sets of mutation probabilities based on the simple cost model due to Böttcher et al. (2010) and the empiric cost model developed here. It is noteworthy that smaller mutation probabilities seem to have much less a detrimental effect than larger ones (compare eq. (12.10)). Since comparisons are difficult to make this way we plot all medians in one common diagram (Figure 12.8) omitting all data except for the medians for the sake of readability. We caution the reader to infer too much from tiny differences in the mean values. Too small differences are likely not to be statistically significant.

In Figure 12.8 we have the median run times in milliseconds for different values of c in the mutation probability $p(n) = c/n$ plotted over the length of the bit string n. We observe that the values between $c = 1/16$ and $c = 2$ form a cluster of very similar run times with $c = 1/4$ being fastest and $c = 2$ being slowest within this efficient cluster. This includes the sets of c-values that are computed depending on n. We notice that there is indeed no advantage for either of them. If at all, smaller mutation probabilities are to be preferred and the standard choice, $p(n) = 1/n$ does pretty well, too. It is worth mentioning that most small mutation probabilities do remarkably well. With larger mutation probabilities, things start to change. When increasing the mutation probability beyond $2/n$ run time increases considerably as can be seen in the plots for

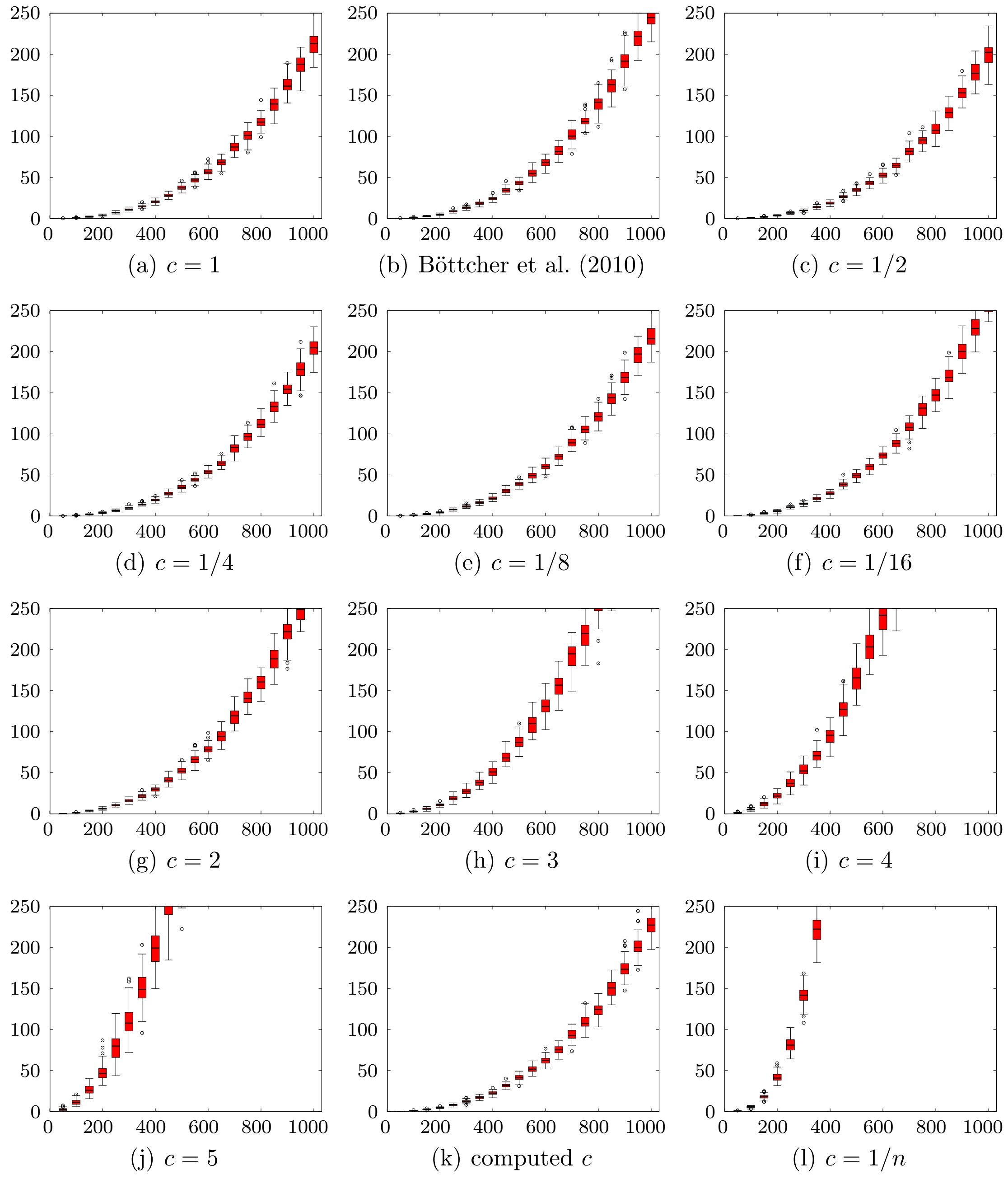


Figure 12.7.: Run times of the (1+1) EA for LeadingOnes for different values of c in the mutation probability $p(n) = c/n$, data from 100 runs. The plots show run times in milliseconds over n.

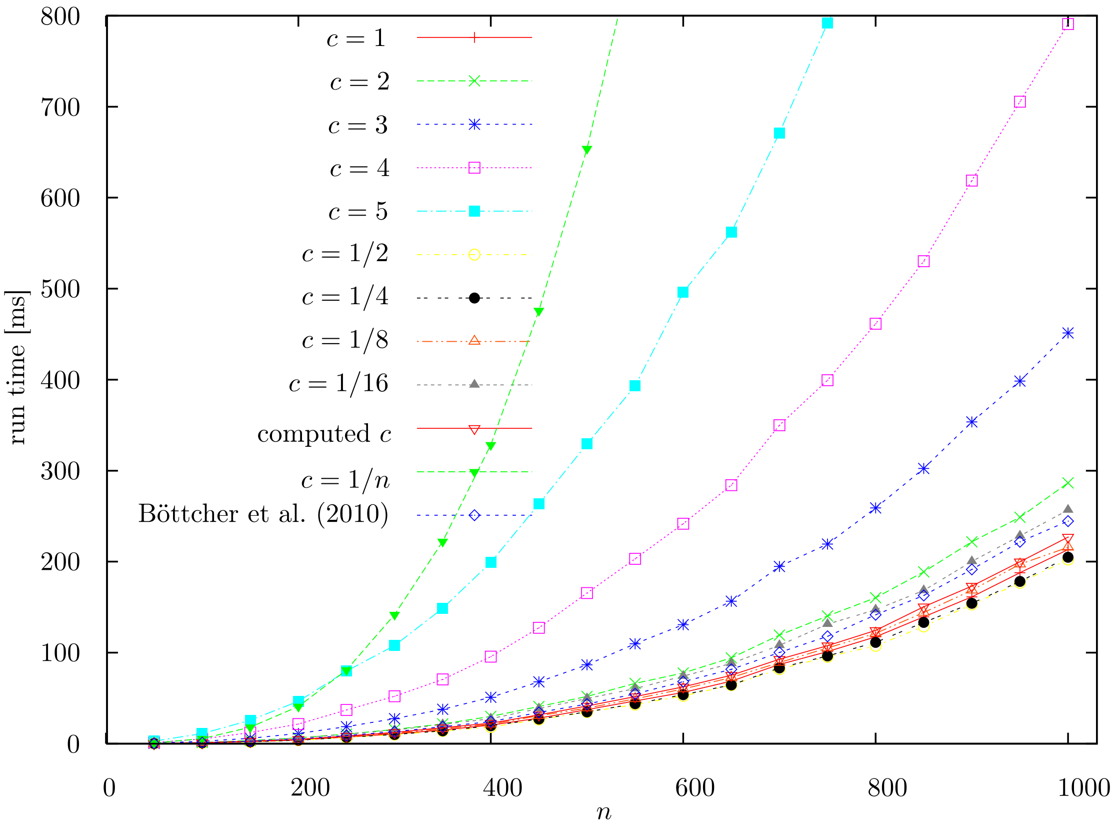


Figure 12.8.: Median run times of the (1+1) EA for LeadingOnes for different values of c in the mutation probability $p(n) = c/n$, data from 100 runs.

$p(n) = c/n$ for $c \in \{3, 4, 5\}$. On the other extreme, the theoretically smallest efficient mutation probability $p(n) = 1/n^2$ turns out to be not efficient at all. Moreover, we point out that with respect to actual run time the result by Böttcher et al. (2010) points in exactly the wrong direction. For LeadingOnes, increasing the mutation probability above the standard choice $p(n) = 1/n$ is a particularly bad idea. Decreasing it is a by far safer choice. The good news is that the performance of the (1+1) EA is quite robust with respect to changes in the mutation probability. The actual run time is not greatly affected as long as some care is taken. Only the extremely small mutation probability $p(n) = 1/n^2$ (corresponding to $c = 1/n$) yields a really bad performance. Given the efficient implementation of mutations it is not surprising that smaller mutation probabilities hurt less than larger ones. Note, however, that all these findings apply to LeadingOnes only, a function where mutations of single bits are sufficient (and even optimal) for optimization. Thus, in the following section, we consider two more example functions where we relax these requirements in two steps.

12.3.6. Other Fitness Functions

All results derived here (as well as in Böttcher et al. (2010)) apply to the (1+1) EA on LeadingOnes, only. Of course, LeadingOnes is a very special function that has a structure that allows for this kind of very precise analysis. It is interesting to find out if the findings for LeadingOnes apply to other functions, too, even if the analytical proofs do not carry over. We restrict our attention in the following to two other well known example functions. Note, that both functions share with LeadingOnes that a single function evaluation is not a complex operation and, thus, the average times for single function evaluations are similar.

The first function we consider here is OneMax (see Definition 4.1 on page 29). Note, that the run time for an evaluation of OneMax is slightly larger than for LeadingOnes since one needs to see all n bits in order to count the number of 1-bits. For computing LeadingOnes(x), it suffices to look at the first LeadingOnes(x) + 1 bits, i. e., stop at the left-most 0-bit.

The function OneMax shares with LeadingOnes the property that mutations of single bits are sufficient to find the optimum efficiently. The expected number of function evaluations is $\Theta(n \log n)$ for any mutation probability $p(n) = c/n$ where $c \in \mathbb{R}_0^+$ is a constant. We perform the same experiments for the same values of c as we did for LeadingOnes and report them in the same way. The single plots for each mutation probability are given in Figure 12.9 in form of box-and-whisker plots.

The first striking difference to LeadingOnes is the clearly increased variance and the larger number of outliers. Both could be expected as it is well known that the run time for LeadingOnes is very much concentrated around the expected value (Droste et al. 2002). For OneMax, this is not the case in this extreme way. The other plots look roughly similar to the corresponding plots for LeadingOnes (but with smaller overall run times, of course). In order to take a closer look we again consider a plot that contains the median values, only (Figure 12.10).

Similar to the results for LeadingOnes, the mutation probabilities $p(n) = c/n$ with $1/16 \leq c \leq 2$ form a band of good performance. Within this, again, differentiation makes hardly any sense. We see again the tendency that decreasing the mutation probability below $1/n$ hurts less than increasing it beyond $1/n$. And, again, the standard choice $p(n) = 1/n$ leads to good performance. A noteworthy difference in comparison to LeadingOnes is the performance when the mutation probability $p(n) = 1/n^2$ (corresponding to $c = 1/n$) is employed. It is comparable in performance to setting $c = 3$ and thus much more efficient than it is for LeadingOnes. We conclude that even very small mutation probabilities like $p(n) = 1/n^2$ (where with probability $(1 - 1/n^2)^n \approx e^{-1/n} \approx 1 - 1/n$ no bit flips at all (Lemma B.8 and B.9)) may lead to efficient optimization since iterations without mutating bits are computationally cheap and small mutation probabilities increase the probability of single bit mutations and decrease the probability of mutations where several bits flip simultaneously. For OneMax, this is favorable. One may speculate that if the simultaneous mutation of some bits is needed things change. In order to investigate that we consider a third example function, Jump$_k$ (Droste et al. 2002).

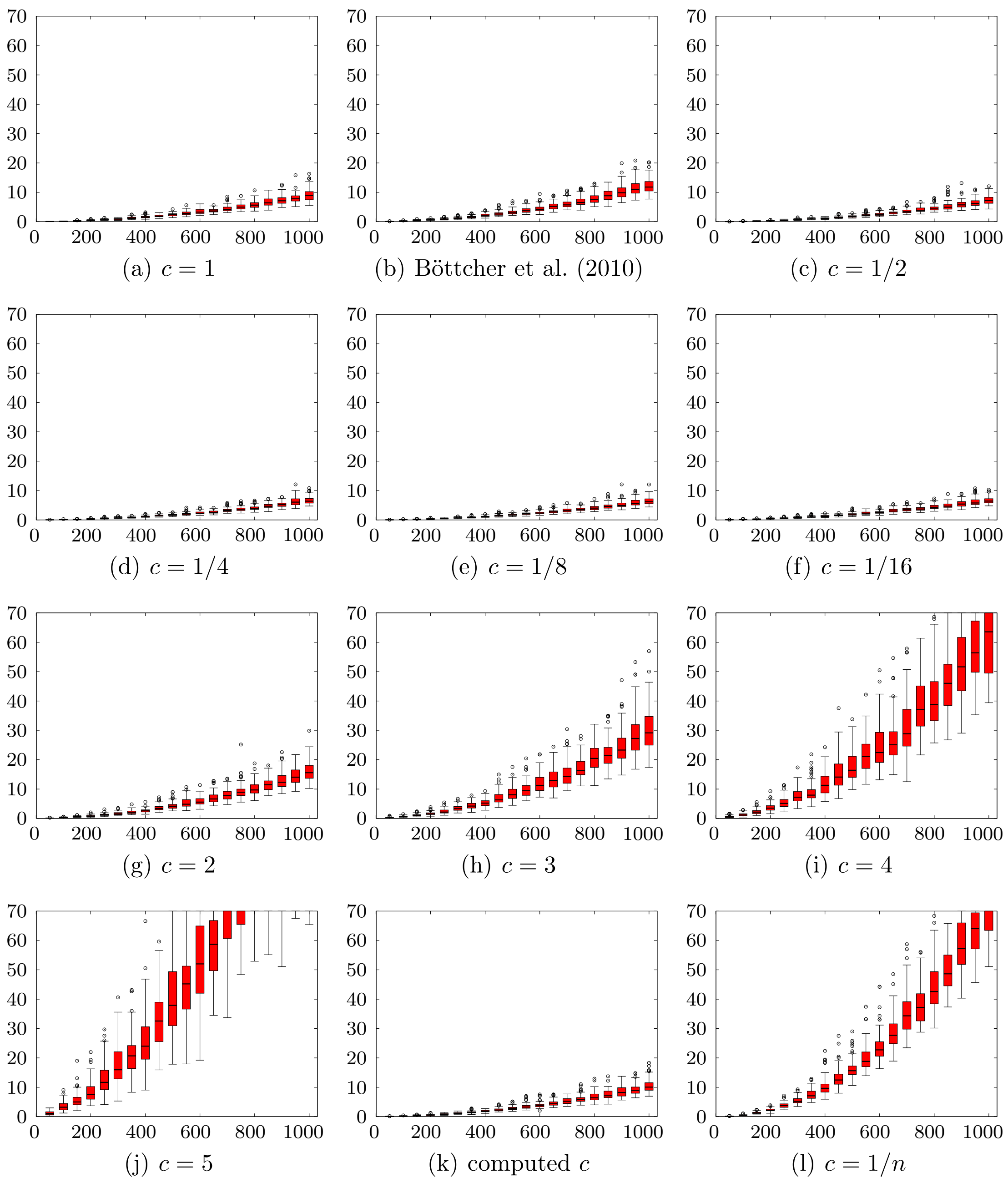


Figure 12.9.: Run times of the (1+1) EA for OneMax for different values of c in the mutation probability $p(n) = c/n$, data from 100 runs. The plots show run times in milliseconds over n.

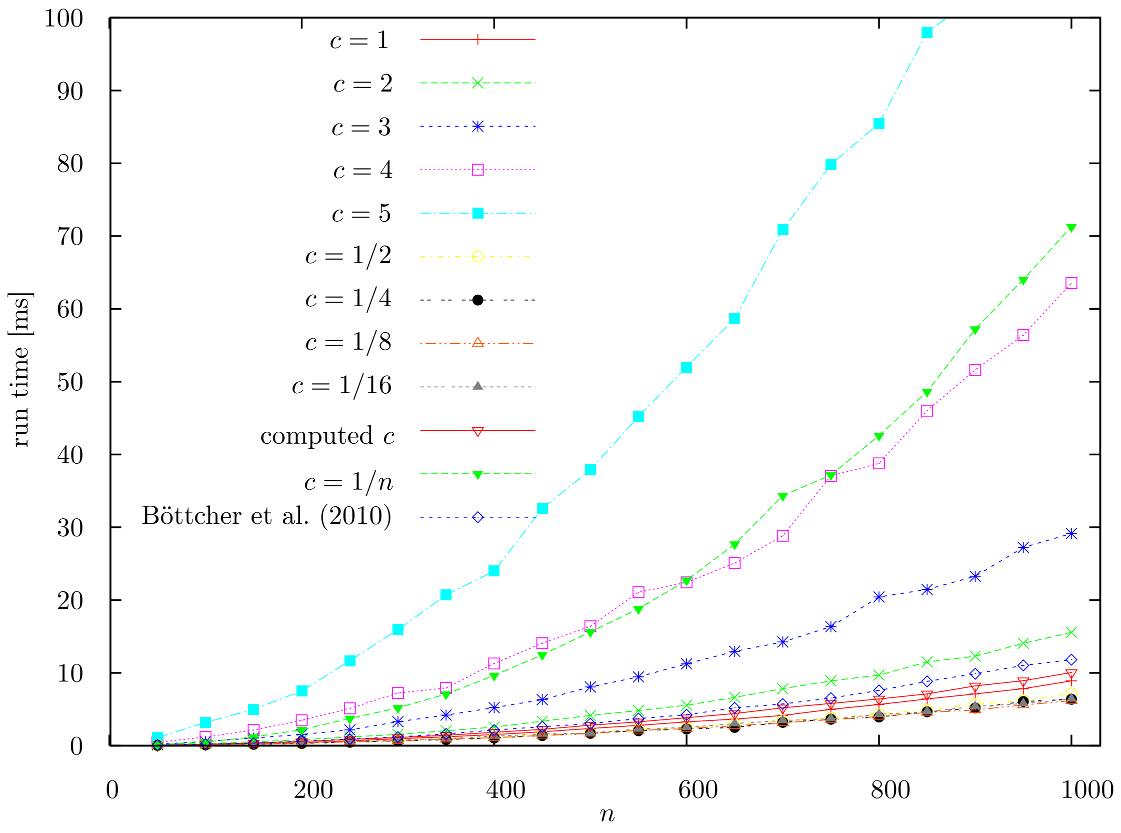


Figure 12.10.: Median run times of the (1+1) EA for ONEMAX for different values of c in the mutation probability $p(n) = c/n$, data from 100 runs.

Definition 12.4. *For $n \in \mathbb{N}$, $x \in \{0,1\}^n$, $k \in \{1, 2, \dots, n\}$, the function* $\text{JUMP}_k \colon \{0,1\}^n \to \mathbb{R}$ *is defined by*

$$\text{JUMP}_k(x) = \begin{cases} |x|_0 & \text{if } n - k < |x|_1 < n, \\ k + |x|_1 & \text{otherwise.} \end{cases}$$

Typically, one considers JUMP_k for small values of k. In these cases the function is very similar to ONEMAX. Only if the number of 1-bits is between $n - k$ and n the function value is very small. Thus, bit strings with exactly $n - k$ 1-bits are easy to locate. From there a direct jump to the unique global optimum, the all-ones bit string, is needed. For the (1+1) EA with mutation probability $p(n) = 1/n$ this dominates the expected number of function evaluation that is $\Theta(n^k + n \log n)$ for all $k \in \{1, 2, \dots, n\}$ (Droste et al. 2002).

We set $k = 2$ and consider JUMP_2 here, only. With this setting, for the (1+1) EA with mutation probability $p(n) = 1/n$, the expected number of function evaluations needed and sufficient for optimization of JUMP_2 equals $\Theta(n^2)$ and is thus asymptotically equal to LeadingOnes. However, a run of the (1+1) EA on Jump_2 will be similar to a run on OneMax. Only if some x with $\text{OneMax}(x) = n - 2$ is reached things change. At that point a mutation of the two remaining 0-bits is needed to find the global optimum. If mutation probability $p(n)$ is used, this mutation has probability $p(n)^2(1-p(n))^{n-2}$. The reciprocal of this mutation probability is the expected waiting time for this mutation and dominates the expected run time. This term becomes minimal for $p(n) = 2/(n-2)$. Thus, we should expect to see good performance when using mutation probability $p(n) = 2/n$, in particular since this mutation probability does not slow down the (1+1) EA on the OneMax-part of the function too much. For the very small mutation probability $p(n) = 1/n^2$ (corresponding to $c = 1/n$) the expected waiting time becomes $\approx n^4$ and we can expect to see dramatically increased run times. As we did for the other two functions we first present the run times in separate box-and-whisker plots (see Figure 12.11).

Things look considerably different for Jump_2 in comparison to LeadingOnes and OneMax. First of all, variance in the run times is even larger than for OneMax. This could be expected since here the run time largely depends on the waiting time for one single event. This implies a much larger variation in comparison to LeadingOnes and OneMax, where the run time is the sum of the waiting times for many such mostly independent events. Second, already in these small plots we can recognize a much increased run time when the mutation probability is decreased. Already for $p(n) = 1/(4n)$ (corresponding to $c = 1/4$) the run times seem to be clearly larger. We consider this in some more detail in Figure 12.12 where only the medians are plotted.

We notice that setting the mutation probability small for Jump_2 is a very bad idea. The mutation probabilities corresponding to $c = 1/n$ and $c = 1/16$ are clearly the two worst choices considered. For the mutation probabilities with $c \in \{1/2, 1/4, 1/8\}$ it is very interesting to note that they pair up with larger values: $c = 1/2$ with $c = 3$, $c = 1/4$ with $c = 4$, and $c = 1/8$ with $c = 5$. We see that even for Jump_2 where a mutation probability of $2/n$ seems to be indicated setting the mutation probability larger is not a good idea. Thus, increasing the mutation probabilities seems to be in general more dangerous with respect to the run time of the (1+1) EA than decreasing them. Considering the other mutation probabilities we see no clear advantage for any of them. In particular, the choice $p(n) = 2/n$ is not superior. Taking into account what we have learned this is easy to explain. Setting the mutation probability slightly smaller than $2/n$ decreases the probability of mutating more than two bits (expensive mutations that are likely to be useless) while slightly increasing the probability for mutations mutating single bits (also expensive, but useful in the OneMax-phase of the optimization) and for mutations mutating no bits at all (cheap mutations).

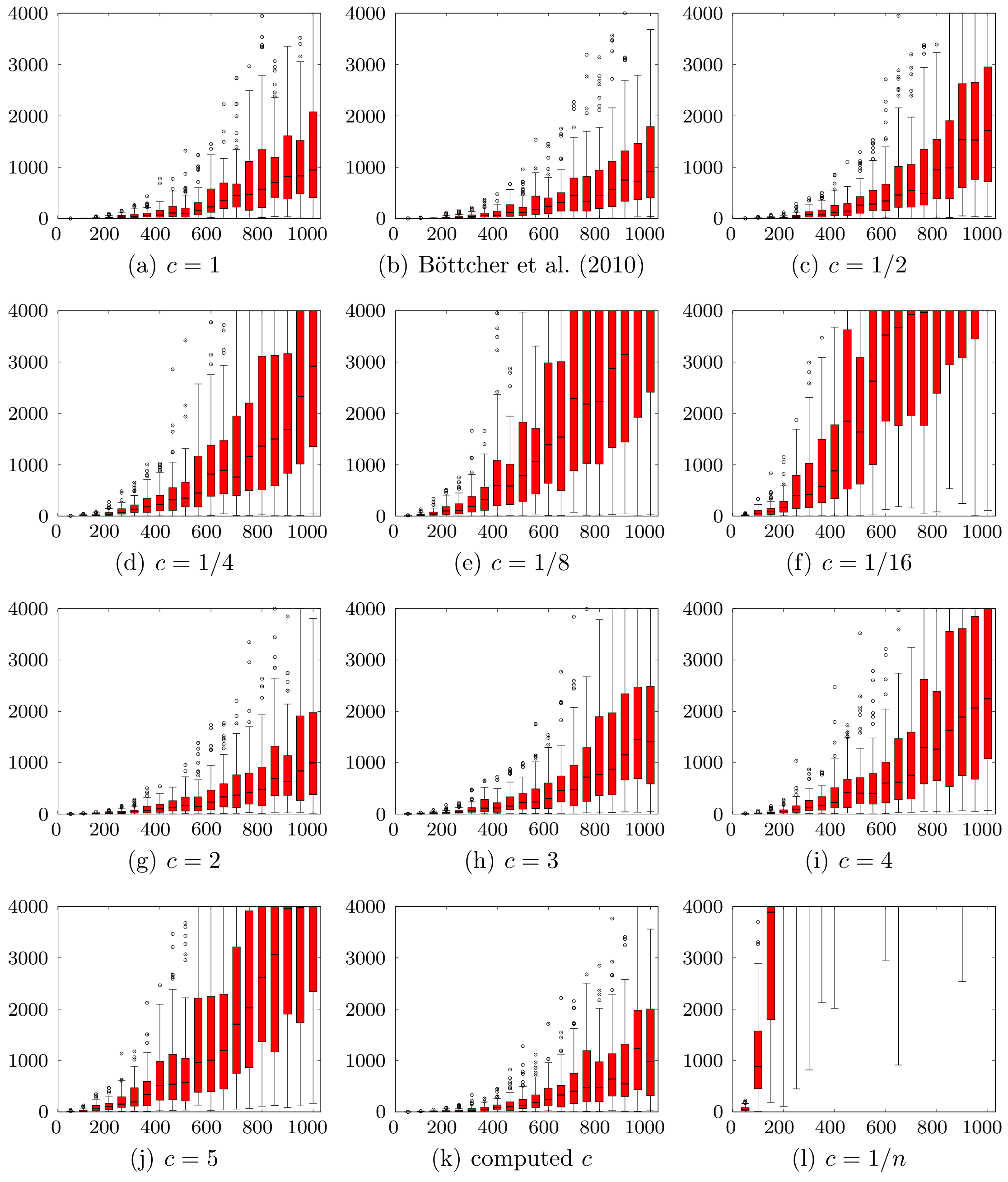


Figure 12.11.: Run times of the (1+1) EA for $\text{J}\scriptstyle{\text{UMP}}_2$ for different values of c in the mutation probability $p(n) = c/n$, data from 100 runs. The plots show run times in milliseconds over n.

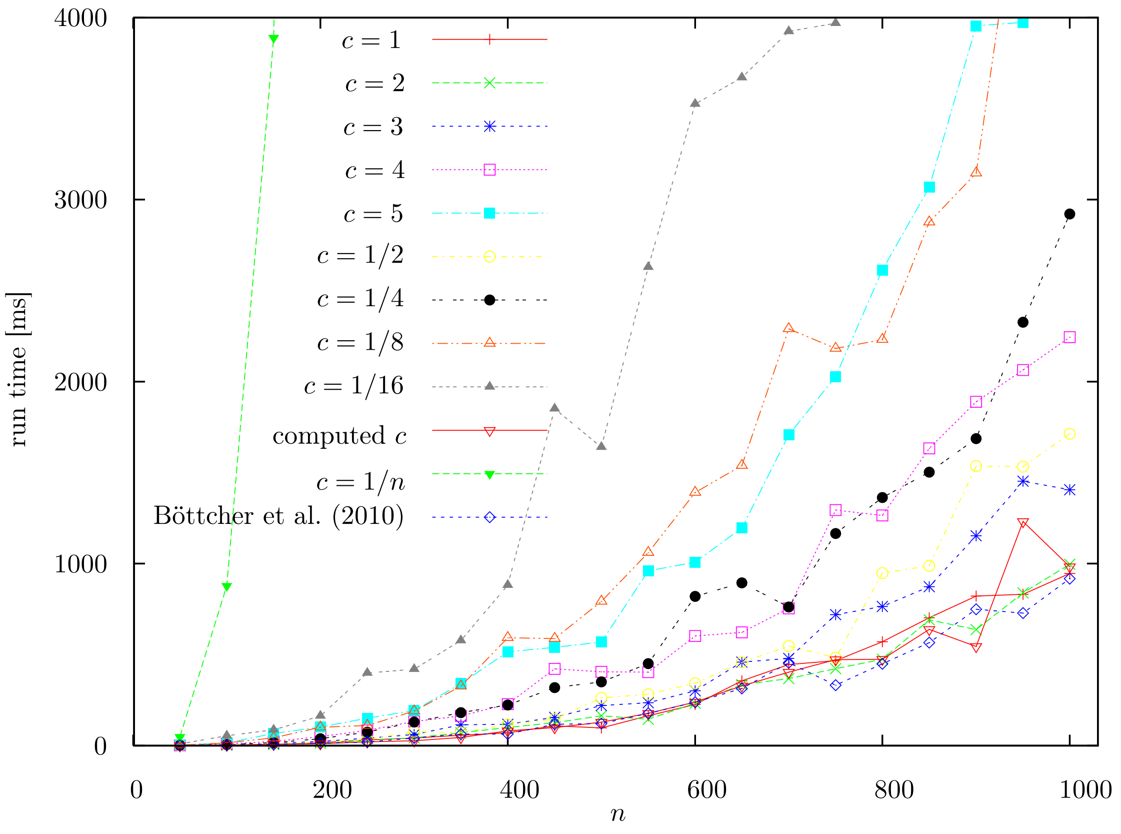


Figure 12.12.: Median run times of the (1+1) EA for J_{UMP_2} for different values of c in the mutation probability $p(n) = c/n$, data from 100 runs.

Part V.
Epilogue

13. Conclusions and Directions for Future Work

In this thesis, we have contributed significantly to the theoretical foundations of artificial immune systems. We considered different important algorithmic aspects, that are inherent in artificial immune systems and distinguished this kind of randomized search heuristics from others, like e. g., evolutionary algorithms. Most of our theoretical results were accompanied by experiments. On one hand we studied the practical relevance of our findings. On the other hand we further investigated certain aspects where rigorous results are missing.

We started with an analysis of typical variation operators. In contrast to other randomized search heuristics, artificial immune systems make use of hypermutation operators, implying mutations at high rate. We considered two concrete implementations of this concept, namely inversely fitness-proportional mutation and contiguous hypermutations, that are used in practical algorithms. We pointed out the importance of an appropriate parametrization and showed on which kind of problems these operators excel the standard operator from the field of evolutionary computation. We closed our examinations of this topic by presenting a general result on high mutation probabilities and showed that on the well-known class of monotone functions even constant changes in the mutation probability can have a tremendous effect. This is the first time such a result was presented.

Reviewing what we have learned in that part of this thesis, we observe that hypermutation operators from artificial immune systems can have difficulties in exactly locating global optima. This is in particularly true for inversely fitness-proportional mutation probabilities since here, the probability for only flipping a small number of bits can be very small. One might conjecture that these kind of operators are more appropriate when searching for a robust solution, i. e., they prefer locally optimal solutions situated in a 'good' area of the search space over an isolated single global optimum. It is an open problem to investigate if this is the case.

With respect to contiguous hypermutations we only considered the extreme parametrization. However, while we have seen that on one hand we loose in comparison to standard bit mutations when 1-bit mutations are essential for the optimization process, we gain advantages on problems, where it is necessary to flip several bits simultaneously. This is often the case if one needs to escape from a local optimum. In order to combine both benefits, it might be reasonable to combine contiguous hypermutations with other mutation operators. We remark, that this is exactly what is done in the B-Cell algorithm where contiguous hypermutations were introduced. It is an open problem to find out what such mechanisms can achieve.

The second aspect investigated in this thesis is the concept of aging. We considered two different implementations of aging, one deriving from the field of artificial immune systems and one from the field of evolutionary computation. We again analyzed appropriate parametrizations for these operators and compared their performance when faced with common properties of a search space, namely local optima and plateaus. Based on our findings we proposed a novel aging operator that provably combines the benefits of the two previous ones. Since in the first results obtained about aging, the same effects can be achieved when replacing aging by some restart strategy, we studied what aging can achieve beyond restarts. One of the main findings here is that static pure aging can be sub-divided into an aging strategy and a replacement strategy that preserves a certain degree of diversity with respect to age.

While we have shown that partial restarts can be beneficial for the optimization process, it is an open problem what aging contributes to the success of algorithms used in practice. It is often stated that aging preserves a certain degree of diversity within the collection of search points. However, we have seen in our analysis that this is not completely true. Other mechanisms have to ensure that aging can be successful. Thus, it is an interesting question for future research to further study the interplay between aging and other mechanisms of the practical algorithms.

Moreover, aging is not only used in the context of selection for replacement. In particular, it is also used to control the mutation strength or the size of the collection of search points. It is an interesting open problem to investigate the effects of aging in other parts of an artificial immune system.

In the last part of this thesis, we considered typical cost models from the field of randomized search heuristics, namely counting iterations and counting function evaluations. Since an appropriate cost model needs to reflect the 'real' run time of an algorithm, we pointed out limitations of both models and proposed a new advanced model that incorporates implementation details and results of experiments to address the identified problems. While this new cost model contributes to the discussion how practical relevant theoretical analyses can be executed, it is only a starting point. In order to bridge the gap between theory and practice more work in this direction needs to be done.

Except for the chapter on monotone functions, all our results are about 'artificial' example functions. These functions are often considered when 'new' search heuristics are subject of rigorous analyses. They help to assess the assets and drawbacks of the heuristics under consideration and get a clearer and better founded understanding of their properties. These results are useful but only a first step. Results on classes of functions, typical situations, and combinatorial optimization problems need to be obtained. Moreover, more realistic algorithms should be considered.

Another important aspect of artificial immune systems is the interplay of different mechanisms known from the immune system of vertebrates. Within this thesis we concentrated on specific operators and analyzed their inherent properties in isolation. This is an important first step in order to understand the functionality of such operators. However, in future research this analysis needs to be extended to more advanced analytical frameworks. A first step in this direction was done in Chapter 10 where we established a structured view on aging and analyzed its interplay with different replacement strategies and a more advance variation operator.

Finally, we only considered a limited number of operators used in artificial immune systems, and in particular only clonal selection algorithms. As discussed in the introduction this is only a small part of the whole area of artificial immune systems. Other algorithms need to be investigated and embedded into the more general field of randomized search heuristics. However, the practical relevance of theoretical findings is crucial. Keeping this in mind, the theoretical analysis can significantly add to the field of artificial immune systems by helping to understand how these algorithms actually work and thus, foster the development of powerful new algorithms.

Appendix

A. Nomenclature

$n \in \mathbb{N}$	dimension of the search space
C_t	collection of search points at time t
C_0	initial collection of search points
$\mu \in \mathbb{N}$	size of the collection of search points
$x_i \in \{0,1\}^n$, $i \in \{1,\dots,\mu\}$	i-th search point of the collection of search points
$x_0 \in \{0,1\}^n$	initial search point
$x[i] \in \{0,1\}$, $i \in \{0,\dots,n-1\}$	i-th bit of search point $x \in \{0,1\}^n$
$\lvert x\rvert_1 \in \{0,\dots,n\}$	number of 1-bits in the bit string $x \in \{0,1\}^n$
$\lvert x\rvert_0 \in \{0,\dots,n\}$	number of 0-bits in the bit string $x \in \{0,1\}^n$
$\mathrm{mut}(x) \in \{0,1\}^n$	result of the mutation of $x \in \{0,1\}^n$
$x^+ \in \{0,1\}^n$	result of a variation and a subsequent selection
$\tau_{\max} \in \mathbb{N}$	maximal lifespan in age-based algorithms
$\log n$	logarithm of n to base 2
$\ln n$	logarithm of n to base e
$H(x,y) \in \{0,\dots,n\}$	Hamming distance of two search points $x, y \in \{0,1\}^n$
$T_{A,f} \in \mathbb{N}_0$	optimization time of Algorithm A on some function $f\colon \{0,1\}^n \to \mathbb{R}$
$T_{A,f,C} \in \mathbb{N}_0$	optimization time of Algorithm A on some function $f\colon \{0,1\}^n \to \mathbb{R}$ when started with some deterministic initial collection of search points C
$\mathrm{Prob}(E) \in [0,1]$	probability of event E
$\mathrm{E}(X)$	expectation of the random variable X
$x \circ y$	concatenation of x and y
H_n	the n-th harmonic number
$\mathrm{poly}(n)$	polynomial in n, $n^{O(1)}$

B. Mathematical Tools

B.1. Definitions

Definition B.1 (Box-and-whisker plots (box plots))**.** *Box-and-whisker plots are a form of representing statistical data. They provide the median together with upper and lower quartiles as well as the minimum and maximum value of the data. We use an extended variant where additionally outliers are identified. We generate box-and-whisker plots using the statistical tool R*[1]*.*

*An example for a box-and-whisker plot as used in this thesis is depicted in Figure B.1. The median is drawn as a thick line. Upper and lower quartile form a rectangle (*box*). Two lines attached to this rectangle indicate the extreme values of the data (*whiskers*). The length of these whiskers is at most 1.5 times the so-called interquartile range (IQR), which is defined as the difference of the upper and lower quartile. More extreme points are considered outliers and drawn as circles.*

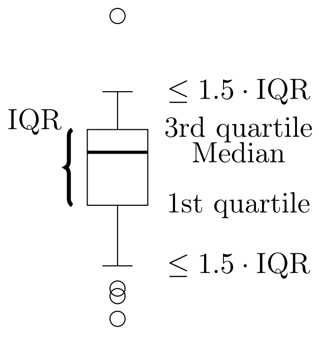


Figure B.1.: Visualization of a box-and-whisker plot.

Definition B.2 (Landau notation (Cormen et al. 2001))**.** *Let $f, g\colon \mathbb{N} \to \mathbb{R}$.*

- *$f(n) = O(g(n)) \Leftrightarrow \exists n_0 \in \mathbb{N}, c \in \mathbb{R}^+ \colon f(n) \leq c \cdot g(n)$, i. e., f does not grow faster than g*
- *$f(n) = \Omega(g(n)) \Leftrightarrow g(n) = O(f(n))$, i. e., f grows at least as fast as g*
- *$f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n))$ and $g(n) = O(f(n))$, i. e., f and g have the same order of growth*

[1]`http://www.r-project.org/`

- $f(n) = o(g(n)) \Leftrightarrow \lim_{n\to\infty} f(n)/g(n) = 0$, *i. e., f grows slower than g*
- $f(n) = \omega(g(n)) \Leftrightarrow g(n) = o(f(n))$, *i. e., f grows faster than g*

Definition B.3 (Conditional Probability (Mitzenmacher and Upfal (2005), Def. 1.4))**.** *The conditional probability that event E event occurs given that event F occurs is*

$$Prob\,(E \mid F) = \frac{Prob\,(E \cap F)}{Prob\,(F)}$$

Definition B.4 (Expectation (Mitzenmacher and Upfal (2005), Def. 2.3))**.** *The expectation of a discrete random variable X, denoted by $E(X)$, is given by*

$$E(X) = \sum_i i Prob\,(X = i)$$

where the summation is over all values in the range of X.

Definition B.5 (Conditional Expectation (Mitzenmacher and Upfal (2005), Def. 2.6))**.**

$$E(Y \mid Z = z) = \sum_y y Prob\,(Y = y \mid Z = z)$$

where the summation is over all y in the range of Y.

B.2. Probability Theory and Other Useful Equations

Lemma B.6 (Harmonic number (Mitzenmacher and Upfal (2005), Lemma 2.10))**.** *The harmonic number $H_n = \sum_{i=1}^{n} 1/i$ satisfies $H(n) \leq \ln(n) + 1$.*

Lemma B.7 (Geometric series (Graham et al. (1994), Sect. 3.2))**.** *For $n \in \mathbb{N}$ and $q \in \mathbb{R}$ with $q \neq 1$, the following holds:*

$$\begin{aligned}\sum_{i=0}^{n} q^i &= \frac{1-q^{n+1}}{1-q} = \frac{q^{n+1}-1}{q-1} \\ \sum_{i=a}^{b} q^i &= \sum_{i=0}^{b} q^i - \sum_{i=0}^{a-1} q^i = \frac{1-q^{b+1}}{1-q} - \frac{1-q^a}{1-q} = \frac{q^a - q^{b+1}}{1-q}\end{aligned}$$

If $q < 1$, additionally the following convergence properties hold:

$$\sum_{i=0}^{\infty} q^i = \frac{1}{1-q} \qquad\qquad \sum_{i=1}^{\infty} q^i = \frac{q}{1-q}$$

Lemma B.8 (Basic estimations (Auger and Doerr (2011), Lemma 1.3))**.** *For all $n \in \mathbb{N}$, the following inequality holds:*

$$\left(1 - \frac{1}{n}\right)^n \leq \frac{1}{e} \leq \left(1 - \frac{1}{n}\right)^{n-1}$$

For $n \geq 2$, we additionally have:

$$\left(1-\frac{1}{n}\right)^n \geq \left(\frac{1}{e}\right)^{\frac{n}{n-1}} = \left(\frac{1}{e}\right)\cdot\left(\frac{1}{e}\right)^{\frac{1}{n-1}} \geq \frac{1}{2e}$$

Lemma B.9 (Bounds on the exponential function (Motwani and Raghavan (1995), Proposition B.3))**.** *For $x \in \mathbb{R}$, the following inequality holds:*

$$\begin{aligned} e^x &\geq 1+x && \text{for } x \in \mathbb{R} \\ e^{-x} &\geq 1-x && \text{for } x \in \mathbb{R} \\ e^x &\leq \frac{1}{1-x} && \text{for } x < 1 \\ 1-e^{-x} &\leq \frac{2x}{1+2x} && \text{for } x \leq 1 \\ 1+2x &\geq e^x && \text{for } x \in [0,1] \end{aligned}$$

Lemma B.10 (Union bound (Mitzenmacher and Upfal (2005), Lemma 1.2))**.** *For any countable, finite sequence of events $E_1, E_2, \ldots,$*

$$Prob\left(\bigcup_{i\geq 1} E_i\right) \leq \sum_{i\geq 1} Prob\left(E_i\right)$$

Lemma B.11 (Law of Total Probability (Mitzenmacher and Upfal (2005), Th. 1.6))**.** *Let $E_1, \ldots, E_n$ be mutually disjoint events in the sample space Ω and let $\bigcup_{i=1}^n E_i = \Omega$. Then*

$$Prob\left(B\right) = \sum_{i=1}^n Prob\left(B \cap E_i\right) = \sum_{i=1}^n Prob\left(B \mid E_i\right)\cdot Prob\left(E_i\right)$$

Lemma B.12 (Linearity of Expectation (Mitzenmacher and Upfal (2005), Th. 2.1))**.** *For any finite collection of discrete random variables $X_1, \ldots, X_n$ with finite expectation*

$$E\left(\sum_{i=1}^n X_i\right) = \sum_{i=1}^n E\left(X_i\right).$$

Lemma B.13 (Binomial distribution (Mitzenmacher and Upfal (2005), Def. 2.5))**.** *A binomial random variable X with parameters n and p, denoted by $B(n,p)$, is defined by the following probability distribution on $j = 0, 1, 2, \ldots, n$:*

$$Prob\left(X = j\right) = \binom{n}{j}\cdot p^j(1-p)^{n-j}$$

Moreover, $E\left(X\right) = pn$.

Lemma B.14 (Geometrical distribution (Mitzenmacher and Upfal (2005), Lemma 2.8)). *A geometric random variable X with parameter p is given by the following probability distribution on $n = 1, 2, \ldots$:*

$$\mathit{Prob}\,(X = n) = (1-p)^{n-1}\,p$$

Moreover, $E(X) = 1/p$.

Lemma B.15 (Stirling's Formula (Mitzenmacher and Upfal (2005), Lemma 7.3)). *For $m > 0$,*

$$\sqrt{2\pi m}\cdot\left(\frac{m}{e}\right)^m \le m! \le 2\sqrt{2\pi m}\left(\frac{m}{e}\right)^m$$

Lemma B.16 (Estimation of Binomial Coefficients (Motwani and Raghavan (1995), Proposition B.2)). *For $n, k \in \mathbb{N}_0$ the following holds.*

$$\binom{n}{k} \le \frac{n^k}{k!}$$
$$\binom{n}{k} \le \left(\frac{ne}{k}\right)^k$$
$$\binom{n}{k} \ge \left(\frac{n}{k}\right)^k$$

Lemma B.17 (Coupon Collector (Motwani and Raghavan (1995), Ch. 3.6)). *Let T denote the number of coupons obtained until all n types of coupons are present for the first time.*

1. $E(T) = n\ln n + O(n)$
2. $\forall\beta > 1 : \mathit{Prob}\,(T > \beta n\ln n) \le n^{-(\beta-1)}$
3. $\forall c \in \mathbb{R} : \mathit{Prob}\,(T > n\ln n + cn) = 1 - e^{-e^{-c}}$

Lemma B.18 (Gambler's Ruin Problem (Feller (1968), Ch. XIV)). *Consider a gambler who wins or loses a dollar with probabilities p and q, respectively. Let q_z be the probability of the gambler's ultimate ruin and p_z the probability of his winning when starting with initial capital z. His adversary starts with initial capital $a - z$. If $p \neq q$ we have*

$$q_z = \frac{(q/p)^a - (q/p)^z}{(q/p)^a - 1}$$

and

$$1 - q_z = \frac{(q/p)^z - 1}{(q/p)^a - 1}.$$

Lemma B.19. *The following statements can be proved by simple inductions.*

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$
$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

Lemma B.20 (Markov inequality (Mitzenmacher and Upfal (2005), Th. 3.1)). *Let X be a non-negative random variable. Then for all $a > 0$*

$$\mathit{Prob}\,(X \geq aE(X)) \leq \frac{1}{a}$$

Lemma B.21 (Chernoff bounds (Mitzenmacher and Upfal (2005), Ch. 4)). *Let $X_1, \ldots, X_n$ be independent Poisson trials such that $\mathit{Prob}\,(X_i) = p_i$. Let $X = \sum_{i=1}^n X_i$ and $\mu = E(X)$. Then the following Chernoff bounds hold.*

- *for any $\delta > 0$*
$$\mathit{Prob}\,(X \geq (1+\delta)\mu) < \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu$$
- *for $0 < \delta \leq 1$*
$$\mathit{Prob}\,(X \geq (1+\delta)\mu) \leq e^{-\frac{\mu\delta^2}{3}}$$
- *for $0 < \delta < 1$*
$$\mathit{Prob}\,(X \leq (1-\delta)\mu) \leq \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^\mu$$
- *for $0 < \delta < 1$*
$$\mathit{Prob}\,(X \leq (1-\delta)\mu) \leq e^{-\frac{\mu\delta^2}{2}}$$

Lemma B.22 (Chernoff bound for moderately independent random variables (p. 13 in Auger and Doerr (2011))). *Let $X_1, \ldots, X_n$ be arbitrary binary random variables. Let $X_1^*, \ldots, X_n^*$ be binary random variables that are mutually independent and such that for all i, X_i^* is independent of $X_1, \ldots, X_{i-1}$. Assume that for all i and all $x_1, \ldots, x_{i-1} \in \{0, 1\}$,*

$$\mathit{Prob}\,(X_i = 1 \mid X_1 = x_1, \ldots, X_{i-1} = x_{i-1}) \leq \mathit{Prob}\,(X_i^* = 1)\,.$$

Then for all $k \geq 0$, we have

$$\mathit{Prob}\left(\sum_{i=1}^n X_i > k\right) \leq \mathit{Prob}\left(\sum_{i=1}^n X_i^* > k\right).$$

The latter term can be bounded by Chernoff bounds for independent random variables.

B.3. Methods from the Theory of Randomized Search Heuristics

Theorem B.23 (Multiplicative Drift (Doerr et al. 2010b)). *Let $\{Z^{(t)}\}_{t \in \mathbb{N}_0}$ be random variables describing a Markov process over a finite state space $S \subseteq \mathbb{R}$. Let T be the random variable that denotes the earliest point in time $t \in \mathbb{N}_0$ such that $Z^{(t)} = 0$.*

If there exist $\delta, c_{\min}, c_{\max} > 0$ such that

1. *$E\big(Z^{(t)} - Z^{(t+1)} \mid Z^{(t)}\big) \geq \delta Z^{(t)}$ and*

2. *$c_{\min} \leq Z^{(t)} \leq c_{\max}$,*

for all $t < T$, then

$$E(T) \leq \frac{2}{\delta} \cdot \ln\left(1 + \frac{c_{\max}}{c_{\min}}\right)$$

Theorem B.24 (Simplified Drift Theorem (Oliveto and Witt 2010))**.** *Let X_t, $t \geq 0$, be the random variables describing a Markov process over a finite state space $S \subseteq \mathbb{R}_0^+$ and denote $\Delta_t(i) := (X_{t+1} - X_t \mid X_t = i)$ for $i \in S$ and $t \geq 0$. Suppose there exist an interval $[a, b]$ in the state space, two constants $\delta, \varepsilon > 0$ and, possibly depending on $I := b - a$, a function $r(I)$ satisfying $1 \leq r(I) = o(I/\log(I))$ such that for all $t \geq 0$ the following two conditions hold:*

1. *$E(\Delta_t(i)) \geq \varepsilon$ for $a < i < b$,*

2. *$Prob\,(\Delta_t(i) \leq -j) \leq \frac{r(I)}{(1+\delta)^j}$ for $i > a$ and $j \in \mathbb{N}_0$.*

Then there is a constant $\kappa > 0$ such that for the time $T^* := \min\{t \geq 0 : X_t \leq a \mid X_0 \geq b\}$ it holds $Prob\,\big(T^* \leq 2^{\kappa I/r(I)}\big) = 2^{-\Omega(I/r(I))}$.

Bibliography

E. Aarts, J. Korst, and W. Michiels (2005). Simulated annealing. In E. K. Burke and G. Kendall, editors, *Search Methodologies*, chapter 7, pages 187—210. Springer.

D. A. Ackley (1987). *A Connectionist Machine for Genetic Hillclimbing.* Kluwer Academic Publishers.

U. Aickelin and S. Cayzer (2002). The danger theory and its application to artificial immune systems. In J. Timmis and P. J. Bentley, editors, *Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS 2002)*, pages 141–148. University of Kent at Canterbury.

U. Aickelin, P. J. Bentley, S. Cayzer, J. Kim, and J. McLeod (2003). Danger theory: The link between AIS and IDS. In J. Timmis, P. J. Bentley, and E. Hart, editors, *Proceedings of the 2nd International Conference on Artificial Immune Systems (ICARIS 2003)*, volume 2787 of *Lecture Notes in Computer Science*, pages 147–155. Springer.

N. Alon and J. H. Spencer (2000). *The Probabilistic Method.* Wiley, 2nd edition.

C. A. C. António (2006). A hierarchical genetic algorithm with age structure for multimodal optimal design of hybrid composites. *Structural and Multidisciplinary Optimization*, 31(4):280–294.

A. Auger and B. Doerr, editors (2011). *Theory of Randomized Search Heuristics.* World Scientific Review.

T. Bäck, Á. E. Eiben, and N. A. L. van der Vaart (2000). An empirical study on GAs “without parameters”. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. M. Guervós, and H.-P. Schwefel, editors, *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature (PPSN VI)*, volume 1917 of *Lecture Notes in Computer Science*, pages 315–324. Springer.

T. Bartz-Beielstein (2006). *Experimental Research in Evolutionary Computation - The New Experimentalism.* Natural Computing Series. Springer.

H. Bernardino and H. Barbosa (2009). Artificial immune systems for optimization. In R. Chiong, editor, *Nature-Inspired Algorithms for Optimisation*, volume 193 of *Studies in Computational Intelligence*, pages 389–411. Springer.

H. Bersini (1992). Immune network and adaptive control. In F. J. Varela and P. Bourgine, editors, *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life (ECAL 1991)*, pages 217–226. MIT Press.

H. Bersini and F. J. Varela (1991a). Hints for adaptive problem solving gleaned from immune networks. In H.-P. Schwefel and R. Männer, editors, *Proceedings of the 1st International Conference on Parallel Problem Solving from Nature (PPSN I)*, volume 496 of *Lecture Notes in Computer Science*, pages 343–354. Springer.

H. Bersini and F. J. Varela (1991b). The immune recruitment mechanism: A selective evolutionary strategy. In R. Belew and L. Booker, editors, *Proceedings of the 4th International Conference on Genetic Algorithms (ICGA 1991)*, pages 520–526. Morgan Kaufmann.

S. Böttcher, B. Doerr, and F. Neumann (2010). Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In R. Schaefer, C. Cotta, J. Kołodziej, and G. Rudolph, editors, *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature (PPSN XI)*, volume 6238 of *Lecture Notes in Computer Science*, pages 1–10. Springer.

J. Brownlee (2007). Clonal selection algorithms. Technical Report 070209A, Swinburne University of Technology, Victoria, Australia.

F. M. Burnet (1959). *The Clonal Selection Theory of Acquired Immunity.* Cambridge University Press.

M. Castrogiovanni, G. Nicosia, and R. Rascunà (2007). Experimental analysis of the aging operator for static and dynamic optimisation problems. In B. Apolloni, R. J. Howlett, and L. C. Jain, editors, *Proceedings of the 11th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES 2007)*, volume 4694 of *Lecture Notes in Computer Science*, pages 804–811. Springer.

J. Cervantes and C. R. Stephens (2008). Rank based variation operators for genetic algorithms. In M. Keijzer, editor, *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO 2008)*, pages 905–912. ACM Press.

E. Clark, A. Hone, and J. Timmis (2005). A Markov chain model of the B-Cell algorithm. In C. Jacob, M. L. Pilat, P. J. Bentley, and J. Timmis, editors, *Proceedings of the 4th International Conference on Artificial Immune Systems (ICARIS 2005)*, volume 3627 of *Lecture Notes in Computer Science*, pages 318–330. Springer.

E. B. Clark (2008). *A Framework for Modelling Stochastic Optimisation Algorithms With Markov Chains.* PhD thesis, University of York.

C. A. Coello Coello and N. C. Cortés (2005). Solving multiobjective optimization problems using an artificial immune system. *Genetic Programming and Evolvable Machines*, 6(2):163–190.

D. E. Cooke and J. E. Hunt (1995). Recognising promoter sequences using an artificial immune system. In C. J. Rawlings, D. A. Clark, R. B. Altman, L. Hunter, T. Lengauer, and S. J. Wodak, editors, *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology (ISMB 1995)*, pages 89–97. AAAI.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein (2001). *Introduction to Algorithms*. MIT Press, 2nd edition.

V. Cutello and G. Nicosia (2002a). An immunological approach to combinatorial optimization problems. In F. J. Garijo, J. C. Riquelme, and M. Toro, editors, *Proceedings of the 8th Ibero-American Conference on Artificial Intelligence (IBERAMIA 2002)*, volume 2527 of *Lecture Notes in Artificial Intelligence*, pages 361–370. Springer.

V. Cutello and G. Nicosia (2002b). Multiple learning using immune algorithms. In *Proceedings of the 4th International Conference on Recent Advances in Soft Computing (RASC 2002)*, pages 102–107.

V. Cutello, G. Nicosia, and M. Pavone (2003). A hybrid immune algorithm with information gain for the graph coloring problem. In E. Cantú-Paz, J. A. Foster, K. Deb, L. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. K. Standish, G. Kendall, S. W. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. A. Dowsland, N. Jonoska, and J. F. Miller, editors, *Proceedings of the 5th Annual Conference on Genetic and Evolutionary Computation (GECCO 2003)*, volume 2723 of *Lecture Notes in Computer Science*, pages 171–182. Springer.

V. Cutello, G. Nicosia, and M. Pavone (2004a). Exploring the capability of immune algorithms: A characterization of hypermutation operators. In G. Nicosia, V. Cutello, P. J. Bentley, and J. Timmis, editors, *Proceedings of the 3rd International Conference on Artificial Immune Systems (ICARIS 2004)*, volume 3239 of *Lecture Notes in Computer Science*, pages 263–276. Springer.

V. Cutello, G. Nicosia, and M. Pavone (2004b). An immune algorithm with hypermacromutations for the Dill's 2d hydrophobic-hydrophilic model. In *Proceedings of the 6th IEEE Congress on Evolutionary Computation (CEC 2004)*, pages 1074–1080. IEEE Press.

V. Cutello, G. Morelli, G. Nicosia, and M. Pavone (2005a). Immune algorithms with aging operators for the string folding problem and the protein folding problem. In G. R. Raidl and J. Gottlieb, editors, *Proceedings of the 5th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2005)*, volume 3448 of *Lecture Notes in Computer Science*, pages 80–90. Springer.

V. Cutello, G. Narzisi, G. Nicosia, and M. Pavone (2005b). Clonal selection algorithms: A comparative case study using effective mutation potentials. In C. Jacob, M. L. Pilat, P. J. Bentley, and J. Timmis, editors, *Proceedings of the 4rd International Conference on Artificial Immune Systems (ICARIS 2005)*, volume 3627 of *Lecture Notes in Computer Science*, pages 13–28. Springer.

V. Cutello, D. Lee, S. Leone, G. Nicosia, and M. Pavone (2006a). Clonal selection algorithm with dynamic population size for bimodal search spaces. In L. Jiao, L. Wang, X. Gao, J. Liu, and F. Wu, editors, *Proceedings of the 2nd International Conference* *on Advances in Natural Computation (ICNC 2006)*, volume 4221 of *Lecture Notes in Computer Science*, pages 949–958. Springer.

V. Cutello, D. Lee, G. Nicosia, M. Pavone, and I. Prizzi (2006b). Aligning multiple protein sequences by hybrid clonal selection algorithm with insert-remove-gaps and blockshuffling operators. In H. Bersini and J. Carneiro, editors, *Proceedings of the 5th International Conference on Artificial Immune Systems (ICARIS 2006)*, volume 4163 of *Lecture Notes in Computer Science*, pages 321–334. Springer.

V. Cutello, G. Nicosia, and M. Pavone (2007a). An immune algorithm with stochastic aging and Kullback entropy for the chromatic number problem. *Journal of Combinatorial Optimization*, 14(1):9–33.

V. Cutello, G. Nicosia, M. Pavone, and J. Timmis (2007b). An immune algorithm for protein structure prediction on lattice models. *IEEE Transactions on Evolutionary Computation*, 11(1):101–117.

V. Cutello, G. Nicosia, M. Romeo, and P. S. Oliveto (2007c). On the convergence of immune algorithms. In *Proceedings of the IEEE Symposium on Foundations of Computational Intelligence (FOCI 2007)*, pages 409–415. IEEE Press.

V. Cutello, G. Nicosia, M. Pavone, and G. Stracquadanio (2010). An information-theoretic approach for clonal selection algorithms. In E. Hart, C. McEwan, J. Timmis, and A. Hone, editors, *Proceedings of the 9th International Conference on Artificial Immune Systems (ICARIS 2010)*, volume 6209 of *Lecture Notes in Computer Science*, pages 144–157. Springer.

D. Dasgupta, editor (1998). *Artificial Immune Systems and Their Applications.* Springer.

D. Dasgupta (2007). Artificial immune systems: A bibliography. Technical Report CS-07-004, Computer Science Department, The University of Memphis, USA. Available online: http://ais.cs.memphis.edu/files/papers/.

D. Dasgupta and L. F. Niño (2008). *Immunological Computation: Theory and Applications.* Auerbach.

L. N. de Castro and J. Timmis (2002a). *Artificial Immune Systems: A New Computational Intelligence Approach.* Springer.

L. N. de Castro and J. Timmis (2002b). An artificial immune network for multimodal function optimization. In *Proceedings of the 4th IEEE Congress on Evolutionary Computation (CEC 2002)*, pages 699–704. IEEE Press.

L. N. de Castro and F. J. Von Zuben (2000). An evolutionary immune network for data clustering. In *Proceedings of the 6th Brazilian Symposium on Neural Networks*, pages 84–89. IEEE Press.

L. N. de Castro and F. J. Von Zuben (2002a). aiNet: An artificial immune network for data analysis. In H. A. Abbass, C. S. Newton, and R. Sarker, editors, *Data Mining: A Heuristic Approach*, chapter XII, pages 231–260. Idea Group Publishing.

L. N. de Castro and F. J. Von Zuben (2002b). Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation*, 6(3):239–251.

K. A. de Jong (2006). *Evolutionary Computation. A Unified Approach.* MIT Press.

P. J. Delves, S. J. Martin, D. R. Burton, and I. M. Roitt (2006). *Roitt's Essential Immunology.* Blackwell Publishing, 11th edition.

B. Doerr and L. A. Goldberg (2010). Adaptive drift analysis. In R. Schaefer, C. Cotta, J. Kołodziej, and G. Rudolph, editors, *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature (PPSN XI)*, volume 6238 of *Lecture Notes in Computer Science*, pages 32–41. Springer.

B. Doerr, N. Hebbinghaus, and F. Neumann (2007). Speeding up evolutionary algorithms through asymmetric mutation operators. *Evolutionary Computation*, 15(4):401–410.

B. Doerr, T. Jansen, D. Sudholt, C. Winzen, and C. Zarges (2010a). Optimizing monotone functions can be difficult. In R. Schaefer, C. Cotta, J. Kołodziej, and G. Rudolph, editors, *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature (PPSN XI)*, volume 6238 of *Lecture Notes in Computer Science*, pages 42–51. Springer.

B. Doerr, D. Johannsen, and C. Winzen (2010b). Multiplicative drift analysis. In J. Branke, editor, *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO 2010)*, pages 1449–1456.

B. Doerr, T. Jansen, D. Sudholt, C. Winzen, and C. Zarges (2011). Mutation rate matters even when optimizing monotone functions. Submitted.

M. Dorigo and T. Stützle (2004). *Ant Colony Optimization.* MIT Press.

S. Droste and D. Wiesmann (2003). On the design of problem-specific evolutionary algorithms. In A. Ghosh and S. Tsutsui, editors, *Advances in Evolutionary Computing*, pages 153–173. Springer.

S. Droste, T. Jansen, and I. Wegener (1998). On the optimization of unimodal functions with the (1+1) evolutionary algorithm. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature (PPSN VII)*, volume 1498 of *Lecture Notes in Computer Science*, pages 13–22. Springer.

S. Droste, T. Jansen, and I. Wegener (2002). On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276(1–2):51–81.

S. Droste, T. Jansen, and I. Wegener (2006). Upper and lower bounds for randomized search heuristics in black-box optimization. *Theory of Computing Systems*, 39:525–544.

M. Elberfeld and J. Textor (2011). Negative selection algorithms on strings with efficient training and linear-time classification. *Theoretical Computer Science*, 412(6):534–542.

J. Farmer, N. Packard, and A. Perelson (1986). The immune system, adaptation, and machine learning. *Physica D*, 2(1–3):187–204.

W. Feller (1968). *An Introduction to Probability Theory and Its Applications. Volume I.* Wiley.

D. Flower and J. Timmis, editors (2007). *In-Silco Immunology.* Springer.

S. Forrest and A. S. Perelson (1991). Genetic algorithms and the immune system. In H.-P. Schwefel and R. Männer, editors, *Proceedings of the 1st International Conference on Parallel Problem Solving from Nature (PPSN I)*, volume 496 of *Lecture Notes in Computer Science*, pages 319–325. Springer.

S. Forrest, B. Javornik, R. E. Smith, and A. S. Perelson (1993). Using genetic algorithms to explore pattern recognition in the immune system. *Evolutionary Computation*, 1 (3):191–211.

S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri (1994). Self-nonself discrimination in a computer. In *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, pages 202–212. IEEE Press.

S. Forrest, S. A. Hofmeyr, and A. Somayaji (1997). Computer immunology. *Communications of the ACM*, 40(10):88–96.

T. Friedrich, N. Hebbinghaus, and F. Neumann (2009a). Comparison of simple diversity mechanisms on plateau functions. *Theoretical Computer Science*, 420(26):2455–2462.

T. Friedrich, P. S. Oliveto, D. Sudholt, and C. Witt (2009b). Analysis of diversity-preserving mechanisms for global exploration. *Evolutionary Computation*, 17(4):455–476.

J. C. Galeano, A. Veloza-Suan, and F. A. González (2005). A comparative analysis of artificial immune network models. In H.-G. Beyer, U.-M. O'Reilly, D. V. Arnold, W. Banzhaf, C. Blum, E. W. Bonabeau, E. Cantu-Paz, D. Dasgupta, K. Deb, J. A. Foster, E. D. de Jong, H. Lipson, X. Llora, S. Mancoridis, M. Pelikan, G. R. Raidl, T. Soule, A. M. Tyrrell, J.-P. Watson, and E. Zitzler, editors, *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation (GECCO 2005)*, pages 361–368. ACM Press.

A. Ghosh, S. Tsutsui, and H. Tanaka (1997). Genetic search with aging of individuals. *International Journal of Knowledge Based Intelligent Engineering Systems*, 1(2):86–103.

F. Glover and M. Laguna (1997). *Tabu Search*. Kluwer Academic Publishers.

R. A. Gonçalves, C. P. de Almeida, M. R. Delgado, E. F. Goldbarg, and M. C. Goldbarg (2007). A cultural immune system for economic load dispatch with non-smooth cost functions. In L. N. de Castro, F. J. Von Zuben, and H. Knidel, editors, *Proceedings of the 6th International Conference on Artificial Immune Systems (ICARIS 2007)*, volume 4628 of *Lecture Notes in Computer Science*, pages 382–394. Springer.

R. L. Graham, D. E. Knuth, and O. Patashnik (1994). *Concrete Mathematics: Foundation for Computer Science*. Addison-Wesley, 2nd edition.

J. Greensmith, U. Aickelin, and S. Cayzer (2005). Introducing dendritic cells as a novel immune-inspired algorithm for anomaly detection. In C. Jacob, M. L. Pilat, P. J. Bentley, and J. Timmis, editors, *Proceedings of the 4th International Conference on Artificial Immune Systems (ICARIS 2005)*, volume 3627 of *Lecture Notes in Computer Science*, pages 153–167. Springer.

J. Greensmith, J. Twycross, and U. Aickelin (2006). Dendritic cells for anomaly detection. In *Proceedings of the 8th IEEE Congress on Evolutionary Computation (CEC 2006)*, pages 664–671. IEEE Press.

G. Harik, E. Cantú-Paz, D. Goldberg, and B. Miller (1999). The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation*, 7(3): 231–253.

E. Hart and J. Timmis (2008). Application areas of AIS: The past, the present and the future. *Applied Soft Computing*, 8(1):191–201.

J. He and X. Yao (2004). A study of drift analysis for estimating computation time of evolutionary algorithms. *Natural Computing*, 3(1):21–35.

J. He, C. R. Reeves, C. Witt, and X. Yao (2007). A note on problem difficulty measures in black-box optimization: Classification, realizations and predictability. *Evolutionary Computation*, 15(4):435–443.

G. W. Hoffmann (1986). A neural network model based on the analogy with the immune system. *Journal of Theoretical Biology*, 122(1):33–67.

J. Horn, D. E. Goldberg, and K. Deb (1994). Long path problems. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Proceedings of the 3rd International Conference on Parallel Problem Solving from Nature (PPSN III)*, volume 866 of *Lecture Notes in Computer Science*, pages 149–158. Springer.

G. S. Hornby (2006). ALPS: the age-layered population structure for reducing the problem of premature convergence. In M. Keijzer, M. Cattolico, D. Arnold, V. Babovic, C. Blum, P. Bosman, M. V. Butz, C. Coello Coello, D. Dasgupta, S. G. Ficici, J. Foster, A. Hernandez-Aguirre, G. Hornby, H. Lipson, P. McMinn, J. Moore, G. Raidl,

F. Rothlauf, C. Ryan, and D. Thierens, editors, *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO 2006)*, pages 815–822. ACM Press.

G. S. Hornby (2009). Steady-state ALPS for real-valued problems. In G. R. Raidl, editor, *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO 2009)*, pages 795–802. ACM Press.

G. S. Hornby (2010). A steady-state version of the age-layered population structure EA. In R. Riolo, U.-M. O'Reilly, and T. McConaghy, editors, *Genetic Programming Theory and Practice VII*, Genetic and Evolutionary Computation, pages 87–102. Springer.

C. Horoba, T. Jansen, and C. Zarges (2009). Maximal age in randomized search heuristics with aging. In G. R. Raidl, editor, *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO 2009)*, pages 803–810. ACM Press.

C. A. Janeway and R. Medzhitov (2002). Innate immune recognition. *Annual Reviews of Immunology*, 20:197—216.

C. A. Janeway, P. Travers, M. Walport, and M. Shlomchik (2005). *Immunobiology: The Immune System in Health and Disease.* Garland Science, 6 edition.

T. Jansen (2001). On classifications of fitness functions. In L. Kallel, B. Naudts, and A. Rogers, editors, *Theoretical aspects of evolutionary computing*, pages 371–386. Springer.

T. Jansen (2002). On the analysis of dynamic restart strategies for evolutionary algorithms. In J. J. Guervós, P. Adamidis, H.-G. Beyer, H.-P. Schwefel, and J.-L. Fernández-Villacañas, editors, *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature (PPSN VII)*, volume 2439 of *Lecture Notes in Computer Science*, pages 33–43. Springer.

T. Jansen (2007). On the brittleness of evolutionary algorithms. In C. R. Stephens, M. Toussaint, D. Whitley, and P. F. Stadler, editors, *Proceedings of the 9th ACM SIGEVO Conference on Foundations of Genetic Algorithms (FOGA 2007)*, volume 4436 of *Lecture Notes in Computer Science*, pages 54–69. Springer.

T. Jansen and D. Sudholt (2010). Analysis of an asymmetric mutation operator. *Evolutionary Computation*, 18(1):1–26.

T. Jansen and I. Wegener (2000). On the choice of the mutation probability for the (1+1) EA. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, *Proceedings of the 6th International Conference on Parallel Problem Solving From Nature (PPSN VI)*, volume 1917 of *Lecture Notes in Computer Science*, pages 89–98. Springer.

T. Jansen and I. Wegener (2001). On the utility of populations. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation (GECCO 2001)*, pages 375–382. Morgan Kaufmann.

T. Jansen and I. Wegener (2002). Evolutionary algorithms — how to cope with plateaus of constant fitness and when to reject strings of the same fitness. *IEEE Transactions on Evolutionary Computation*, 5(6):589–599.

T. Jansen and I. Wegener (2007). A comparison of simulated annealing with a simple evolutionary algorithm on pseudo-Boolean functions of unitation. *Theoretical Computer Science*, 386(1–2):73–93.

T. Jansen and R. P. Wiegand (2004a). The cooperative coevolutionary (1+1) EA. *Evolutionary Computation*, 12(4):405–434.

T. Jansen and R. P. Wiegand (2004b). Bridging the gap between theory and practice. In X. Yao, E. K. Burke, J. A. Lozano, J. Smith, J. J. M. Guervós, J. A. Bullinaria, J. E. Rowe, P. Tiño, A. Kabán, and H.-P. Schwefel, editors, *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature (PPSN VIII)*, volume 3242 of *Lecture Notes in Computer Science*, pages 61–71. Springer.

T. Jansen and C. Zarges (2009a). A theoretical analysis of immune inspired somatic contiguous hypermutations for function optimization. In P. S. Andrews, J. Timmis, N. D. Owens, U. Aickelin, E. Hart, A. Hone, and A. M. Tyrrell, editors, *Proceedings of the 8th International Conference on Artificial Immune Systems (ICARIS 2009)*, volume 5666 of *Lecture Notes in Computer Science*, pages 80–94. Springer.

T. Jansen and C. Zarges (2009b). Comparing different aging operators. In P. S. Andrews, J. Timmis, N. D. Owens, U. Aickelin, E. Hart, A. Hone, and A. M. Tyrrell, editors, *Proceedings of the 8th International Conference on Artificial Immune Systems (ICARIS 2009)*, volume 5666 of *Lecture Notes in Computer Science*, pages 95–108. Springer.

T. Jansen and C. Zarges (2010a). Aging beyond restarts. In J. Branke, editor, *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO 2010)*, pages 705–712. ACM Press.

T. Jansen and C. Zarges (2010b). On the benefits of aging and the importance of details. In E. Hart, C. McEwan, J. Timmis, and A. Hone, editors, *Proceedings of the 9th International Conference on Artificial Immune Systems (ICARIS 2010)*, volume 6209 of *Lecture Notes in Computer Science*, pages 61–74. Springer.

T. Jansen and C. Zarges (2011a). Analyzing different variants of immune inspired somatic contiguous hypermutations. *Theoretical Computer Science*, 412(6):517–533.

T. Jansen and C. Zarges (2011b). On benefits and drawbacks of aging strategies for randomized search heuristics. *Theoretical Computer Science*, 412(6):543–559.

T. Jansen and C. Zarges (2011c). On the role of age diversity for effective aging operators. *Evolutionary Intelligence*, 4(2):99–125.

T. Jansen and C. Zarges (2011d). Analysis of evolutionary algorithms: From computational complexity analysis to algorithm engineering. In H.-G. Beyer and W. B. Langdon, editors, *Proceedings of the 11th ACM SIGEVO Conference on Foundations of Genetic Algorithms (FOGA 2011)*, pages 1–14. ACM Press.

N. Jerne (1974). Towards a network theory of the immune system. *Annals of Immunology*, 125C(1–2):373–389.

Z. Ji and D. Dasgupta (2007). Revisiting negative selection algorithms. *Evolutionary Computation*, 15(2):223–251.

J. Kelsey and J. Timmis (2003). Immune inspired somatic contiguous hypermutation for function optimisation. In E. Cantú-Paz, J. A. Foster, K. Deb, L. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. K. Standish, G. Kendall, S. W. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. A. Dowsland, N. Jonoska, and J. F. Miller, editors, *Proceedings of the 5th Annual Conference on Genetic and Evolutionary Computation (GECCO 2003)*, volume 2723 of *Lecture Notes in Computer Science*, pages 207–218. Springer.

J. O. Kephart (1994). A biologically inspired immune system for computers. In R. A. Brooks and P. Maes, editors, *Artificial Life IV: Proceedings of the Fourth International Workshop on Synthesis and Simulatoin of Living Systems*, pages 130–139. MIT Press.

J.-H. Kim, H.-K. Chae, J.-Y. Jeon, and S.-W. Lee (1996). Identification and control of systems with friction using accelerated evolutionary programming. *IEEE Control Systems Magazine*, 16(4):38–47.

D. Knuth (1969). *The Art of Computer Programming. Volume II.* Addison-Wesley.

N. Kubota and T. Fukuda (1997). Genetic algorithms with age structure. *Soft Computing*, 1(4):155–161.

H. Lamlum, M. Ilyas, A. Rowan, S. Clark, V. Johnson, J. Bell, I. Frayling, J. Efstathiou, K. Pack, S. Payne, R. Roylance, P. Gorman, D. Sheer, K. Neale, R. Phillips, I. Talbot, W. Bodmer, and I. Tomlinson (1999). The type of somatic mutation at APC in familial adenomatous polyposis is determined by the site of the germline mutation: a new facet to Knudson's 'two-hit' hypothesis. *Nature Medicine*, 5(9):1071–1075.

R. E. Langman and M. Cohn (1986). The 'complete' idiotype network is an absurd immune system. *Immunology Today*, 7(4):100–101.

R. E. Langman and M. Cohn (2000). Self-nonself discrimination revisited. Introduction. *Seminars in Immunology*, 12(3):159–162.

P. K. Lehre and X. Yao (2011). On the impact of mutation-selection balance on the runtime of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*. To appear, available online: http://dx.doi.org/10.1109/TEVC.2011.2112665.

M. Liskiewicz and J. Textor (2010). Negative selection algorithms without generating detectors. In J. Branke, editor, *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO 2010)*, pages 1047–1054. ACM Press.

K. Mak and P. S. Lau (2008). Order pickings in an AS/RS with multiple I/O stations using an artificial immune system with aging antibodies. *Engineering Letters*, 16(1): 122–130.

P. Matzinger (1994). Tolerance, danger, and the extended family. *Annual Reviews of Immunology*, 12:991–1045.

Z. Michalewicz (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer.

W. Michiels, E. Aarts, and J. Korst (2007). *Theoretical Aspects of Local Search*. Springer.

M. Mitzenmacher and E. Upfal (2005). *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press.

R. Motwani and P. Raghavan (1995). *Randomized Algorithms*. Cambridge University Press.

F. Neumann and C. Witt (2010). *Bioinspired Computation in Combinatorial Optimization - Algorithms and Their Computational Complexity*. Natural Computing Series. Springer.

P. Oliveto, J. He, and X. Yao (2007). Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results. *International Journal of Automation and Computing*, 4:281–293.

P. S. Oliveto and C. Witt (2010). Simplified drift analysis for proving lower bounds in evolutionary computation. *Algorithmica*, 59(3):369–386.

P. S. Oliveto, P. K. Lehre, and F. Neumann (2009). Theoretical analysis of rank-based mutation: combining exploration and exploitation. In *Proceedings of the 11th IEEE Congress on Evolutionary Computation (CEC 2009)*, pages 1455–1462. IEEE Press.

A. Perelson (1989). Immune network theory. *Immunological Review*, 110(1):5–36.

A. Perelson and G. Oster (1979). Theoretical studies of clonal selection: minimal antibody repertoire size and reliability of self-non-self discrimination. *Journal of Theoretical Biology*, 81(4):645–670.

R. Quick, V. Rayward-Smith, and G. Smith (1998). Fitness distance correlation and ridge functions. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature (PPSN V)*, volume 1498 of *Lecture Notes in Computer Science*, pages 77–86. Springer.

Y. Rabinovich, A. Sinclair, and A. Wigderson (1992). Quadratic dynamical systems. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science (FOCS 1992)*, pages 304–313. IEEE Press.

R. Rojas (1996). *Neural Networks – A Systematic Introduction.* Springer.

G. Rudolph (1996). How mutation and selection solve long-path problems in polynomial expected time. *Evolutionary Computation*, 4(2):195–205.

G. Rudolph (1997). *Convergence Properties of Evolutionary Algorithms.* Kovac.

G. Rudolph and J. Ziegenhirt (1997). Computation time of evolutionary operators. In T. Bäck, D. B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, pages E2.2:1–4. CRC Press.

P. Sanders (2009). Algorithm engineering – an attempt at a definition. In S. Albers, H. Alt, and S. Näher, editors, *Efficient Algorithms – Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday*, volume 5760 of *Lecture Notes in Computer Science*, pages 321–340. Springer.

M. D. Schmidt and H. Lipson (2011). Age-fitness pareto optimization. In R. Riolo, T. McConaghy, and E. Vladislavleva, editors, *Genetic Programming Theory and Practice VIII*, Genetic and Evolutionary Computation, pages 129–146. Springer.

H.-P. Schwefel (1995). *Evolution and Optimum Seeking.* Wiley & Sons.

H.-P. Schwefel and G. Rudolph (1995). Contemporary evolution strategies. In F. Morán, A. Moreno, J. J. M. Guervós, and P. Chacón, editors, *Proceedings of the 3rd European Conference on Artificial Life (ECAL 1995)*, volume 929 of *Lecture Notes in Computer Science*, pages 893–907. Springer.

P. E. Seiden and F. Celada (1992). A model for simulating cognate recognition and response in the immune system. *Journal of Theoretical Biology*, 158(3):329–357.

S. Stepney, R. E. Smith, J. Timmis, A. M. Tyrrell, M. J. Neal, and A. N. Hone (2005). Conceptual frameworks for artificial immune systems. *International Journal of Unconventional Computing*, 1:315—338.

T. Storch (2004). On the choice of the population size. In K. Deb, R. Poli, W. Banzhaf, H.-G. Beyer, E. K. Burke, P. J. Darwen, D. Dasgupta, D. Floreano, J. A. Foster, M. Harman, O. Holland, P. L. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. M. Tyrrell, editors, *Proceedings of the 6th Annual Conference on Genetic and Evolutionary*

Computation (GECCO 2004), volume 3102 of *Lecture Notes in Computer Science*, pages 748–760. Springer.

D. Sudholt (2008). *Computational Complexity of Evolutionary Algorithms, Hybridizations, and Swarm Intelligence*. PhD thesis, Technische Universität Dortmund.

G. Tedesco, J. Twycross, and U. Aickelin (2006). Integrating innate and adaptive immunity for intrusion detection. In H. Bersini and J. Carneiro, editors, *Proceedings of the 5th International Conference on Artificial Immune Systems (ICARIS 2006)*, volume 4163 of *Lecture Notes in Computer Science*, pages 193–202. Springer.

J. Timmis (2006). Challenges for artificial immune systems. In B. Apolloni, M. Marinaro, G. Nicosia, and R. Tagliaferri, editors, *Proceedings of the 16th Italian Workshop on Neural Nets (WIRN 2005) and the International Workshop on Natural and Artificial Immune Systems (NAIS 2005)*, volume 3931 of *Lecture Notes in Computer Science*, pages 355–367. Springer.

J. Timmis (2007). Artificial immune systems – today and tomorrow. *Natural Computing*, 6(1):1–18.

J. Timmis, P. Andrews, N. Owens, and E. Clark (2008a). Immune systems and computation: An interdisciplinary adventure. In C. S. Calude, J. F. Costa, R. Freund, M. Oswald, and G. Rozenberg, editors, *Proceedings of the 7th International Conference on Unconventional Computing (UC 2008)*, volume 5204 of *Lecture Notes in Computer Science*, pages 8–18. Springer.

J. Timmis, P. Andrews, N. Owens, and E. Clark (2008b). An interdisciplinary perspective on artificial immune systems. *Evolutionary Intelligence*, 1(1):5–26.

J. Timmis, A. Hone, T. Stibor, and E. Clark (2008c). Theoretical advances in artificial immune systems. *Theoretical Computer Science*, 403(1):11–32.

J. Twycross and U. Aickelin (2005). Towards a conceptual framework for innate immunity. In C. Jacob, M. L. Pilat, P. J. Bentley, and J. Timmis, editors, *Proceedings of the 4rd International Conference on Artificial Immune Systems (ICARIS 2005)*, volume 3627 of *Lecture Notes in Computer Science*, pages 112–125. Springer.

J. Twycross and U. Aickelin (2006). libtissue - implementing innate immunity. In *Proceedings of the 8th IEEE Congress on Evolutionary Computation (CEC 2006)*, pages 499–506. IEEE Press.

R. E. Vance (2000). Cutting edge commentary: A Copernican revolution? Doubts about the danger theory. *Journal of Immunology*, 165:1725–1728.

M. Villalobos-Arias, C. A. Coello Coello, and O. Hernández-Lerma (2004). Convergence analysis of a multiobjective artificial immune system algorithm. In G. Nicosia, V. Cutello, P. J. Bentley, and J. Timmis, editors, *Proceedings of the 3rd International*

Conference on Artificial Immune Systems (ICARIS 2004), volume 3239 of *Lecture Notes in Computer Science*, pages 226–235. Springer.

I. Wegener (2003). Methods for the analysis of evolutionary algorithms on pseudo-boolean functions. In R. Sarker, M. Mohammadian, and X. Yao, editors, *Evolutionary Optimization*, volume 48 of *International Series in Operations Research & Management Science*, chapter 14, pages 349–369. Kluwer Academic Publishers.

C. Witt (2006). Runtime analysis of the (μ+1) EA on simple pseudo-Boolean functions. *Evolutionary Computation*, 14(1):65–86.

C. Witt (2008). Population size versus runtime of a simple evolutionary algorithm. *Theoretical Computer Science*, 403(1):104–120.

C. Witt (2009). Why standard particle swarm optimisers elude a theoretical runtime analysis. In I. Garibay, T. Jansen, R. P. Wiegand, and A. S. Wu, editors, *Proceedings of the 10th ACM SIGEVO Conference on Foundations of Genetic Algorithms (FOGA 2009)*, pages 13–19. ACM Press.

C. Zarges (2008). Rigorous runtime analysis of inversely fitness proportional mutation rates. In G. Rudolph, T. Jansen, S. Lucas, C. Poloni, and N. Beume, editors, *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature (PPSN X)*, volume 5199 of *Lecture Notes in Computer Science*, pages 112–122. Springer.

C. Zarges (2009). On the utility of the population size for inversely fitness proportional mutation rates. In I. Garibay, T. Jansen, R. P. Wiegand, and A. S. Wu, editors, *Proceedings of the 10th ACM SIGEVO Conference on Foundations of Genetic Algorithms (FOGA 2009)*, pages 39–46. ACM Press.