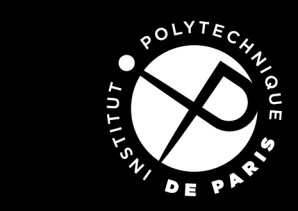
Methods for Tight Analysis of Population-based Evolutionary Algorithms

Denis Antipov



INSTITUT POLYTECHNIQUE DE PARIS

ÉCOLE POLYTECHNIQUE

IP PARIS

ITMO UNIVERSITY

NNT : 2020IPPAX069

Thèse de doctorat

Methods for Tight Analysis of Population-based Evolutionary Algorithms

Thèse de doctorat de l'Institut Polytechnique de Paris préparée à l'École polytechnique et à l'Université d'État en technologie de l'information, mécanique et optique de St-Pétersbourg ITMO

École doctorale n°626 L'École Doctorale de l'Institut Polytechnique de Paris (ED IP Paris)
Spécialité de doctorat: Informatique

Thèse présentée et soutenue à Palaiseau, le 5 November 2020, par

DENIS ANTIPOV

Composition du Jury :

Christoph Durr Directeur de recherche, Sorbonne Université and CNRS	Président
Laetitia Jourdan Professeure des Universités, Université de Lille	Rapporteur
Anton Eremeev Directeur de recherche, Omsk Branch of Sobolev Institute of Mathematics (Siberian Branch of Russian Acadey of Sciences)	Rapporteur
Jesse Read Professeur, École polytechnique	Examinateur
Alberto Tonda Chargé de recherche, Institut National de Recherche pour l'Agriculture, l'Alimentation et l'Environnement	Examinateur
Alexander Boukhanovsky Professeur, ITMO University	Examinateur
Maxim Buzdalov Professeur associé, ITMO University	Directeur de thèse
Benjamin Doerr Professeur, École Polytechnique	Directeur de thèse

Title: New Methods for Tight Analysis of Population-based Evolutionary Algorithms

Keywords: Algorithms, Runtime Analysis, Evolutionary Computation

Abstract: Evolutionary Algorithms (EAs for brevity) is a broad class of optimization algorithms which are inspired by the natural evolution. They are often used to solve practical problems which cannot be solved precisely in a reasonable time, since they can find satisfying solutions without spending too much computation resources. The practical efficiency of the EAs is supported by the theory of evolutionary computation which has produced a huge number of impressive results during the last two decades. These results give valuable recommendations on how to set up parameters of algorithms or even propose new EAs.

Theoretical studies mostly observe how simple algorithms optimize model problems. It is hard to analyse more real-world settings, since even the most simple algorithms are often described via highly complicated stochastic processes. In particular, not so much is known about the behavior of the population-based algorithms, while the populations are believed to be essential by most practitioners. The lack of theoretical understanding of how populations work raises a risk that populations are used not in the most effective way.

The existing analysis tools, however, are not suitable to give us a better understanding of populations. Hence, the main aim of this work is to develop new analysis methods which would help to deliver new runtime bounds for evolutionary algorithms and extend our knowledge of the role of populations. We propose the following **analysis methods for the population-based EAs**.

1) The *method of the complete trees* for delivering the lower bounds on the runtime of the population-based EAs.
2) The *method* of the analysis of the *no-drift* processes.
3) The *method* for delivering the precise bounds on the runtime distribution for the EAs on *plateaus*.
4) The *additive drift theorem with tail bounds*.

With these analysis methods we perform a **runtime analysis** of the following algorithms.

1) With the method of the complete trees we derive a tight bound on the runtime of the $(\mu + \lambda)$ EA on ONEMAX. These bounds, in particular, suggest that using parent population size μ which is $O(\log(n))$ does not increase the asymptotical runtime (which can be useful when optimizing noisy functions) and that using offspring population size λ greater than $\max\{\mu, \ln(n)\}$ does not give a significant decrease in the expected number of iterations (hence such large population sizes should be avoided when solving some easy problems).
2) With the method of the analysis of the no-drift processes we analyse the (μ, λ) EA on ONEMAX with the threshold parameter values $\lambda \approx e\mu$. We show that in this setting (where there is almost no drift of the number of the best offspring) the absolute population size plays a significant role and that this regime seems to be the most interesting for the practical application of the (μ, λ) EA.
3) With the method for the analysis of EAs on plateaus we deliver precise estimates on the runtime of the $(1+1)$ EA and $(\lambda \overset{1:1}{+} \lambda)$ EA on PLATEAU_k function, demonstrating that the choice of the mutation operator does not play a significant role when an EA is traversing a plateau.

We also propose a **new crossover-based algorithm** with non-trivial offspring population — the *heavy-tailed* $(1 + (\lambda, \lambda))$ *GA*. Our analysis of this algorithm on ONEMAX, LEADINGONES and JUMP_k functions reveals the efficiency of the random parameter choices from a power-law distribution. While on JUMP_k this random parameter choice gives us a one-size-fits-all algorithm which relieves us from choosing the optimal static parameters (which depend on function parameter k, probably unknown in advance), on ONEMAX we observed a runtime which is better than the runtime for the best static parameter choice. On the LEADINGONES (with a help of the developed additive drift theorem) we showed that the asymptotical runtime is the same for any static or dynamic choice of parameters and is the same as for the most standard mutation-based algorithms.

Institut Polytechnique de Paris
91120 Palaiseau, France

Titre: Nouvelles Méthodes d'analyse précise des algorithmes évolutifs basés sur la population

Mots clés: Algorithmique, Analyse de la Complexité, Calcul Évolutif

Résumé: Les algorithmes évolutifs (AÉs pour la brièveté) sont une large classe d'algorithmes d'optimisation qui sont inspirés par l'évolution naturelle. Ils sont souvent utilisés pour résoudre des problèmes pratiques qui ne peuvent pas être résolus avec précision dans un délai raisonnable, car ils peuvent trouver des solutions satisfaisantes sans dépenser trop de ressources de calcul. L'efficacité pratique des AÉ est soutenue par la théorie du calcul évolutif, qui a produit un grand nombre de résultats impressionnants au cours des deux dernières décennies. Ces résultats donnent de précieuses recommandations sur la façon de configurer les paramètres des algorithmes ou même proposent des nouvaux AÉs.

Les études théoriques observent principalement comment des algorithmes simples optimisent des problèmes de modèle. Il est difficile d'analyser problèmes du monde réel, car même les algorithmes les plus simples sont souvent décrits via des processus stochastiques très compliqués. En particulier, on en sait peu sur le comportement des algorithmes basés sur des populations. En même temps les populations sont considérées comme essentielles par la plupart des praticiens. Le manque de compréhension théorique du fonctionnement des populations soulève le risque que les populations ne sont pas utilisées de la manière la plus efficace.

Les outils d'analyse existants ne sont cependant pas adaptés pour donner une meilleure compréhension des populations. Ainsi, l'objectif principal de cette thèse est de développer de nouvelles méthodes d'analyse qui permettraient de fournir de nouvelles limites d'exécution pour les algorithmes évolutifs et d'étendre nos connaissances sur le rôle des populations. Nous proposons les **méthodes d'analyse pour les AÉs basées sur la population**.

1) La *méthode des arbres complets* pour fournir les limites inférieures au temps d'exécution des AÉs basées sur la population.
2) La *méthode* de l'analyse des processus *sans dérive*.
3) La *méthode* pour fournir les limites précises sur la distribution des temps d'exécution pour les AÉs sur *plateaux*.
4) Le *théorème de dérive additif avec limites de la distribution*.

Avec ces méthodes d'analyse, nous effectuons une **analyse d'exécution** des algorithmes suivants.

1) Avec la méthode des arbres complets, nous dérivons une limite étroite sur l'exécution de l'AÉ $(\mu+\lambda)$ sur le problème OneMax. Ces limites, en particulier, suggèrent que l'utilisation d'une taille de la population parente μ qui est $O(\log(n))$ n'augmente pas le temps d'exécution asymptotique (ce qui peut être utile lors de l'optimisation des fonctions bruyantes) et que l'utilisation de la taille de la population de la progéniture λ supérieur à $\max\{\mu, \ln(n)\}$ ne donne pas une diminution significative du nombre d'itérations prévu (par conséquent, de telles tailles de population devraient être évitées lors de la résolution de problèmes faciles).
2) Avec la méthode d'analyse des processus sans dérive, nous analysons l'AÉ (μ, λ) sur OneMax avec les valeurs de paramètre de seuil $\lambda \approx e\mu$. Nous montrons que dans ce cadre (où il n'y a quasiment pas de dérive du nombre de meilleurs descendants) la taille absolue de la population joue un rôle significatif et que ce régime semble être le plus intéressant pour l'application pratique de l'AÉ (μ, λ).
3) Avec la méthode d'analyse des AÉs sur plateaux nous délivrons des estimations précises sur l'exécution de l'AÉ $(1+1)$ et l'AÉ $(\lambda \overset{1:1}{+} \lambda)$ sur Plateau$_k$, démontrant que le choix de l'opérateur de mutation ne joue pas un rôle significatif lorsqu'un les AÉs traverse un plateau.

Nous proposons également un **nouvel algorithme basé sur le croisement** avec une population de descendants non triviale — le *queue-lourde l'algorithme génétique* $(1 + (\lambda, \lambda))$. Notre analyse de cet algorithme sur les fonctions OneMax, LeadingOnes et Jump$_k$ révèle l'efficacité des choix de paramètres aléatoires à partir d'une loi de puissance. Alors que sur Jump$_k$ ce choix de paramètre aléatoire nous donne un algorithme universel, ce qui nous dispense de choisir les paramètres statiques optimaux (qui dépendent du paramètre de fonction *k*, probablement inconnu à l'avance), sur OneMax nous observons un temps d'exécution meilleur que le temps d'exécution pour le meilleur choix de paramètre statique. Sur le fonction LeadingOnes (avec l'aide du théorème de dérive additif avec limites de la distribution), nous montrons que le temps d'exécution asymptotique est le même pour tout choix statique ou dynamique des paramètres et est le même que pour les algorithmes basés sur des mutations les plus standards.

Institut Polytechnique de Paris
91120 Palaiseau, France

Тема диссертации: Методы точного анализа популяционных эволюционных алгоритмов

Ключевые слова: Алгоритмы, Анализ времени работы, Эволюционные вычисления

Аннотация: Эволюционные алгоритмы (ЭА) — это широкий класс алгоритмов оптимизации, основанных на принципах естественной эволюцией. Они часто используются для решения практических задач, которые нельзя решить точно за разумное время, поскольку они могут найти достаточно хорошие решения, не затрачивая черезмерно много вычислительных ресурсов. Практическая эффективность ЭА также подтверждается теорией эволюционных вычислений, которая получила большое число значимых результатов за последние два десятилетия. Эти результаты дают ценные рекомендации по настройке параметров алгоритмов или даже предлагают новые ЭА.

Теоретические исследования в основном изучают, как простые алгоритмы оптимизируют модельные задачи. Анализировать более близкие к реальному миру задачи намного сложнее, так как даже самые простые алгоритмы часто описываются с помощью очень сложных стохастических процессов. В частности, довольно мало изучено поведение популяционных алгоритмов, в то время как на практике нетривиальные популяции считаются неотъемлимой составной частью эволюционных алгоритмов. Отсутствие теоретического понимания того, как работают популяции, повышает риск того, что популяции используются не самым эффективным образом.

Однако, существующие инструменты теоретического анализа не могут дать лучшее понимание природы популяций. Следовательно, основная цель этой работы состоит в разработке новых методов анализа для оценки времени работы для популяционных эволюционных алгоритмов и для расширения наших знаний о роли популяций. В данной работе были предложены следующие **методы анализа популяционных ЭА**.

1) *метод полных деревьев* для получения нижних границ на время работы популяционных ЭА.
2) *метод* анализа процессов *без явного сноса*.
3) *метод* для точной оценки распределения времени работы для ЭА на *плато*.
4) *Теорема об аддитивном сносе с оценками на концентрацию*.

С помощью этих методов анализа был выполнен **анализ времени работы** следующих алгоритмов.

1) С помощью метода полных деревьев были получены асимптотически точные оценки на время работы $(\mu + \lambda)$-ЭА на OneMax. Эти оценки, в частности, предполагают, что использование размера популяции родителей μ, равного $O(\log(n))$, не увеличивает асимптотическое время работы (что может быть полезно при оптимизации функций в присутствии шума), и что использование размера популяции потомков λ больше, чем $\max\{\mu, \ln(n)\}$, не дает значительного уменьшения ожидаемого числа итераций (следовательно, следует избегать таких больших размеров популяции при решении простых задач).
2) С помощью метода анализа процессов без явного сноса был проведен анализ (μ, λ)-ЭА на OneMax с граничными значениями параметров $\lambda \approx e\mu$. Было показано, что в этой ситуации (где практически отсутствует снос случайной величины, равной числу лучших потомков в поколении) абсолютный размер популяции играет значительную роль, и что этот режим является наиболее интересным для практического применения (μ, λ)-ЭА.
3) С помощью метода анализа ЭА на плато были получены точные оценки времени работы $(1 + 1)$-ЭА и эволюционного алгоритма $(\lambda \overset{1:1}{+} \lambda)$ на функции Plateau_k, показавшие, что выбор оператора мутации не играет существенной роли, когда ЭА находится на плато.

Также был предложен **новый алгоритм, использующий оператор скрещивания** с нетривиальной популяцией потомков — *генетический алгоритм* $(1 + (\lambda, \lambda))$ *с тяжелым хвостом* (то есть использующий распределения, не сконцентрированные у малых значений). Проведенный анализ этого алгоритма на функциях OneMax, LeadingOnes и Jump_k показал эффективность случайного выбора параметров из степенного распределения. В то время как на Jump_k этот случайный выбор параметров дает нам универсальный алгоритм, который избавляет нас от выбора оптимальных статических параметров (которые зависят от параметра k, вероятно, неизвестного заранее), на OneMax наблюдается время работы, которое лучше, чем время работы алгоритма с наилучшими статическими параметрами. На LeadingOnes (с помощью предложенной теоремы об аддитивном сносе) было показано, что асимптотическое время работы одинаково для любого статического или динамического выбора параметров, причем оно такое же, как для большинства стандартных алгоритмов, основанных на мутациях.

Institut Polytechnique de Paris
91120 Palaiseau, France

Contents

Introduction

Relevance.

Evolutionary algorithms (EAs for brevity) are a class of the random search heuristics which are based on the principles of the natural evolution. Most of the algorithms which belong to this class are described by the following scheme.

1) First EAs create an initial population of the potential solutions, which are usually random points of the search space.
2) Then they evaluate the individuals of the population by calculating their *fitness*, that is, the value of the optimized function in them.
3) Then the algorithm chooses parents from the current population using some selection mechanism and apply different variation operators (*mutation* and *crossover*) to them. The resulting set of the new potential solutions is called the *offspring population*. It might complement the parent population (in case of *elitist* algorithms) or totally replaces it (in case of *non-elitist* algorithms). Then the algorithm removes the less fit individuals and returns to the second step.
4) The algorithm is terminated as soon as some stopping criterion is met (e.g., if a sufficiently good solution is found or if the algorithm is run out of the computation resources).

EAs are often not capable of finding the exact optimal solution in a polynomial time (e.g., when solving an NP-hard problem), however they can find a sufficiently good solution in sufficiently short time. For this reason EAs are often used to solve such practical problems as making a schedule for the ships to pass through a channel [103], or optimization of space trajectories [93]. Also there is a huge variety of additional heuristics which let the user to tailor an EA for his problem. At the same most of the theoretical studies consider only the most simple algorithms on toy problems. The reason of this gap between theory and practice is that on the one hand EAs are easy to implement and tailor for a particular problem, while on the other hand the stochastic processes which describe them are extremely complex and require non-trivial analysis methods.

Nevertheless, the theory is an important part of the field of the evolutionary computation, since it has given a lot of valuable recommendations on how to use the EAs in practice (e.g., recommendation to use a standard bit mutation with rate $\frac{1}{n}$ [67]) and introduced some new algorithms into the field (e.g., the Fast genetic algorithm and the $(1 + (\lambda, \lambda))$ GA [34]).

Most of such results were obtained via the runtime analysis. By the *runtime* of an algorithm we mean not the clock time, but the number of fitness evaluations or the number of iterations which the algorithm makes before finding the optimum. This measure is used in all theoretical studies for several reasons. Fist, it is independent of the details of the algorithm implementation and of the hardware which runs the algorithm. And second, the most computation time of an EA run is usually spent on the fitness evaluations, while the intermediate calculations take only a small part of the resources. The main goal of the theoretical runtime analysis is to deliver the estimates on the algorithm runtime considering it as a random variable. Namely, the aim is to find its expectation, variance or tail bounds.

There is a rich toolbox for the theoretical runtime analysis, which includes

- Markov's chains,
- drift analysis,

- fitness levels techniques,
- family trees

and other methods and techniques. Unfortunately, most of them have some own disadvantages which make them hard to apply for the non-trivial EAs, such as population-based EAs, or they do not give enough insights about the algorithm behavior, which prevent us from delivering recommendations on the practical application of EAs. In particular, Markov's chains have a very complicated structure for the population-based algorithms, drift analysis requires to design a complicated potential function which has to take into account the whole population, fitness levels techniques are either not precise enough, or they do not give much insights. The method of family trees has only a narrow application field, since so far it was used only for the analysis of the $(\mu + 1)$ EA.

Populations are widely used when solving practical problems with EAs. However, the lack of the theoretical understanding of the concept of populations does not let us use them in the most effective ways (e.g., it is not clear how to choose the population size). The root of this problem is in the absence of the methods for the runtime analysis of the population-based EAs. Hence, the topic of the new methods of the tight runtime analysis of the population-based EAs is *relevant*.

State of the art.

Markov's chains are a classic tool from the probability theory which is widely used in the field of the theory of evolutionary computation [15, 9]. However, they are mostly used for the analysis of simple processes or for the analysis of simple components of a process.

Drift analysis is a set of theorems which allow to estimate the runtime T until the random process reaches some value through the expected change of the process in a unit of time. These methods originate from the negative drift by Hajek [78]. A big step in the drift analysis was the development of the additive drift theorem by He and Yao [80], which then lead to its various modifications. However, all these theorems have two disadvantages. First, they all require to design a potential function defined on all possible states of the analysed algorithm. It is possible for simple algorithms like the $(1 + 1)$ EA, however it becomes too complicated when we analyse a population-based EA which has much more possible states. Second, most of these theorems only give the bounds on the expectation of T and only few of them give some tail bounds. The most detailed survey on the drift analysis can be found in [101].

Fitness levels techniques are the methods which are based on splitting the search space into sets of points with the same fitness [17]. These methods often consider the EAs with a large level of abstraction. Although such approach makes the application field of these methods significantly wider, it also prevents us from getting insights on the algorithm behavior and reduces the precision.

Family trees were developed for the proof of lower bounds on runtime [138]. They made it possible to obtain the lower bounds fot the $(\mu + 1)$ EA, but since then it was not used outside this scope (in particular, for the analysis of the algorithms with non-trivial offspring population).

The theoretical toolbox also contains some problem-specific methods like the multi-branching processes [96] or tools which have not shown their true potential yet (mostly because they require some additional analysis before the application). An example for the latter group of methods is the stochastic domination [28].

From the analysis of the above-mentioned results, the following conclusions can be derived:

- the existing methods are either too complicated for the analysis of the population-based EAs or do not give sufficient insights on the EAs behavior;
- it is possible to develop new methods for the analysis of the more complicated processes (like the population-based EAs) based on the existing methods.

The **aim** of this work is to increase the efficiency of the population-based evolutionary algorithms

To achieve this aim, the following **tasks** are defined:

1) develop the methods for the analysis of the population-based EAs;
2) apply the developed methods to the population-based EAs;
3) distill the recommendations on the practical use of the population-based EAs and develop new efficient algorithms.

The **object of the study** are bio-inspired optimization methods (evolutionary algorithms in particular), which use non-trivial populations.

The **subject of the study** is the methods of the precise runtime bounds delivery for such algorithms.

Principal statements of the thesis:

1) the *method of the complete trees* for delivering the lower bounds on the runtime of the population-based EAs;
2) the *method* of the analysis of the *no-drift* processes;
3) the *method* for delivering the precise bounds on the runtime distribution for the EAs on *plateaus*;
4) the *additive drift theorem with tail bounds*;
5) the *algorithm* which is based on the $(1 + (\lambda, \lambda))$ GA and which is more efficient that the standard EAs both on unimodal and multimodal problems.

The **scientific novelty** is as follows:

1) the method of complete trees (in contrast with the method of family trees) considers also search points which potentially could be created by the algorithm with a different parent selection mechanism;
2) the method of the no-drift processes transforms a no-drift process to a process with a drift
3) the method of the runtime analysis of EAs on plateaus is the first to analyse the spectrum of the transition matrix in order to get precise bounds;
4) the new drift theorem uses the original negative drift theorem in order to obtain the tail bounds;
5) the proposed algorithm is the first crossover-based algorithm which uses the fast mutation operator.

Research methodology and methods. This thesis uses methods of discrete mathematics, probability theory, linear algebra and experiment design and analysis.

Soundness and correctness of scientific statements, conclusions and proofs obtained in the thesis are confirmed by formalized problem settings, formal proofs of correctness and efficiency of proposed models, methods and algorithms, as well as by the experimental results.

The **theoretical significance** of the thesis is that, with the use of proposed models, methods and algorithms, the following statements were proven:

- the method of complete trees significantly expands the possibilities of the family trees making it possible to analyse algorithms with non-trivial offspring population;

- the method of analysis of the no-drift processes allows to apply a huge variety of the drift theorems to the drift theorems to the processes which have no significant drift;
- the method of the analysis of EAs on plateaus gives a better understanding of the behavior of the EAs on plateaus and shows that the runtime of unary algorithms has only a weak dependency on the choice of the mutation operator;
- additive drift theorem with tail bounds allows to obtain the estimates of the runtime distribution under the additive drift, while it does not require to prove complicated properties of a process;
- the proposed $(1 + (\lambda, \lambda))$ GA with heavy-tailed choice of the parameters shows how effective it can be to combine the two previous theoretical results, namely the $(1 + (\lambda, \lambda))$ GA and the fast mutation operator.

The **practical significance** of the thesis is that application of the proposed methods to the runtime analysis of population-based EAs allowed to distill the following recommendations on how to use EAs for solving practical problems:

- for the $(\mu + \lambda)$ EA using population sizes which are $O(\log(n))$ does not increase the asymptotical runtime, which can be useful when optimizing noisy functions;
- for the $(\mu + \lambda)$ EA using the population size greater than $\max\{\mu, \ln(n)\}$ does not give a significant decrease in the expected number of iterations, hence such large population sizes should be avoided when solving some easy problems;
- for the (μ, λ) EA one should use the population sizes such that $\lambda \approx e\mu$ and the algorithm behavior can be adjusted by choosing the right absolute population size;
- when solving problems with plateaus, it is hard to increase the efficiency by varying the mutation operator, hence one should use crossover-based algorithms for such problems;
- when solving problems without any information on their structure, one should use the proposed modifications of the $(1 + (\lambda, \lambda))$ GA, since they release the algorithm from parameter tuning.

The results of the thesis **were used** in the following projects:

- project No. 17-71-20178 “Methods for development of efficient evolutionary algorithms”, years 2017–2020, supported by Russian Science Foundation;
- project “Methods, models and technologies of the artificial intelligence in bioinformatics, social media, cyber-physical, biometric and speech systems”, years 2017–2019, supported by the Government of Russian Federation.

The results were also implemented into the educational process of the Faculty of Information Technologies and Programming of ITMO University in course «Genetic and evolutionary computation» of the master’s program «Technologies of software development».

Dissemination. The main results of the thesis were presented at the following venues:

1) Genetic and Evolutionary Computation Conference , GECCO (2018, Kyoto, Japan; 2019, Prague, Czech Republic; 2020, online);
2) Parallel Problem Solving from Nature, PPSN (2018, Coimbra, Portugal; 2020, Leiden, Netherlands);
3) Foundations of Genetic Algorithms (2019, Potsdam, Germany).

Personal contribution. The author has developed the analysis methods, performed the runtime analysis of the EAs with the developed methods, the design of the proposed algorithms and the design of the experimental studies. In papers co-authored with supervisors Maxim Buzdalov and Benjamin Doerr their contribution is in the supervision of the work. The contribution

of Maxim Buzdalov is also in developing the software for running the experiments. In papers co-authored with students Jiefeng Fang, Tangi Hetet, Quentin Yang and Vitalii Karavaev supervised (or co-supervised) by Denis Antipov their contribution is in checking the proof ideas.

Publications. The primary results of this thesis are presented in nine publications, eight of which are published in the conference proceedings indexed in Scopus, two of which are also indexed in Web of Science. Also there is a publication published in a journal indexed in Scopus.

There are also ten other publications related to the design and analysis of evolutionary algorithms, six of which are indexed in Scopus, including two publications which are also indexed in Web of Science.

Chapter 1 Introduction

Evolutionary algorithms (EAs for brevity) are a class of the random search heuristics which are based on the principles of the natural evolution. Most of the algorithms which belong to this class are described by the following scheme.

1) First, EAs create an initial population of the potential solutions which are usually random points of the search space.
2) Then EAs evaluate the individuals of the population by calculating their *fitness*, that is, the value of the optimized function at the individual.
3) Then the algorithm chooses parents from the current population using some selection mechanism and apply variation operators (*mutation* and *crossover*) to the selected parents. The resulting set of the new potential solutions is called the *offspring population*. It might complement the parent population (in case of *elitist* algorithms) or totally replace it (in case of *non-elitist* algorithms). Then the algorithm removes the less fit individuals and returns to the second step.
4) The algorithm is terminated as soon as some stopping criterion is met (e.g., if a sufficiently good solution is found or if the algorithm is run out of the computation resources).

EAs are often not capable of finding the exact optimal solution in a polynomial time (e.g., when solving an NP-hard problem), however they can find a sufficiently good solution in sufficiently short time. For this reason EAs are often used to solve such hard practical problems as making a schedule for ships to pass through a channel [103], or optimization of space trajectories [93]. Also there is a huge variety of additional heuristics which let the user to tailor an EA for his problem. At the same time most of the theoretical studies consider only the most simple algorithms on toy problems. The reason of this gap between theory and practice is that on the one hand EAs are easy to implement and tailor for a particular problem, while on the other hand the stochastic processes which describe them are extremely complex and require non-trivial analysis methods.

Nevertheless, theory is an important part of the field of the evolutionary computation, since it has given a lot of valuable recommendations on how to use the EAs in practice (e.g., recommendation to use a standard bit mutation with rate $\frac{1}{n}$ [67]) and introduced some new algorithms into the field (e.g., the fast genetic algorithm [57] and the $(1 + (\lambda, \lambda))$ GA [34]). On the other hand, theoretical understanding of such an important concept of the EAs as populations is still on an unsatisfying level. The main problem here is that the algorithms with non-trivial populations are described via complex stochastic processes (much more complicated than the algorithms with trivial populations) which are extremely hard to analyse with the existing mathematical tools. The lack of theoretical understanding of populations raises the risk that populations might be used not in the most effective way and encourages the development of new analysis tools to overcome it.

Most of theoretical results were obtained via the runtime analysis. By the *runtime* of an algorithm we mean not the clock time, but the number of fitness evaluations or the number of iterations which the algorithm makes before finding the optimum. This measure is used in most theoretical studies for at least two reasons. Fist, it is independent of the details of the algorithm implementation and of the hardware which runs the algorithm. And second, the most

computation time of an EA run is usually spent on the fitness evaluations, while the intermediate calculations take only a small part of the resources.

The main goal of the theoretical studies is usually to consider some algorithm (or a class of algorithms) optimizing some function (or a class of functions) and to deliver the estimates on its runtime considering it as a random variable. Namely, the aim is to find its expectation, variance or tail bounds.

In the following sections of Chapter 1 we describe the algorithms and functions considered in this work after introducing the notation used in this work. Then we describe the current state of the art of the theory of the evolutionary computation and its existing toolbox. Then we show what is missing in this toolbox and explain the relevance of this work. We conclude the chapter with the collection of mathematical tools which are used in this work.

1.1 Notation

In this section we briefly overview the notation we use to avoid misreading of the results presented in the work.

By the set of natural integers $\mathbb{N}$ we denote the set of positive integers $\{1, 2, \dots\}$ and by $\mathbb{N}_0$ we denote $\mathbb{N} \cup \{0\}$.

We write $[a..b]$ to denote an integer interval including its borders and $(a..b)$ to denote an integer interval excluding its borders. For $a, b \in \mathbb{R}$ the notion $[a..b]$ means $[\lceil a \rceil .. \lfloor b \rfloor]$. For the real-valued intervals we write $[a, b]$ and (a, b) respectively.

For any probability distribution $\mathcal{L}$ and random variable X, we write $X \sim \mathcal{L}$ to indicate that X follows the law $\mathcal{L}$. We denote the Bernoulli law of parameter $p \in [0, 1]$ by $\mathrm{Ber}(p)$ and the binomial law with parameters $n \in \mathbb{N}$ and $p \in [0, 1]$ by $\mathrm{Bin}(n, p)$.

An empty product (i.e. a product over an empty set) is always considered to be 1, an empty sum is always 0. The infimum of an empty set is $+\infty$.

The binomial coefficient $\binom{n}{k}$ is zero, if $k < 0$ or if $k > n$.

We denote the vector of length n that consists only of ones by 1^n and the vector of length n that consists only of zeros by 0^n.

By $\mathbb{1}_A$ we denote a random variable which equals to one if and only if event A occurs and which is equal to zero otherwise.

We follow the common notation and call evolutionary algorithms which use a crossover operator *genetic algorithms* (GAs for brevity).

1.2 Considered EAs and GAs

In this section we describe the algorithms which are considered in this work, namely the $(\mu + \lambda)$ EA, the (μ, λ) EA and the $(1 + (\lambda, \lambda))$ GA. Although these algorithms can be used in different search spaces, in this work we consider optimization of pseudo-Boolean functions, that are, functions which work from the space of bit strings of length n to the set of real numbers

Algorithm 1 – The $(\mu+\lambda)$ EA maximizing a given function $f : \{0,1\}^n \to \mathbb{R}$, with population size μ and offspring population size λ. We do not specify the mutation operator.

Initialization:
Create a population of μ individuals by choosing $x^{(i)} \in \{0,1\}^n$, $1 \leq i \leq \mu$ uniformly at random.
Let the multiset $X^{(0)} := \{x^{(1)}, ..., x^{(\mu)}\}$ be the population at time 0. Let $t := 0$.
Optimization:
while an optimum has not been reached **do**
    $X' := X^{(t)}$
    Mutation phase:
    for $i = 1, \ldots, \lambda$ **do**
        Choose $x \in X^{(t)}$ uniformly at random
        Create x' by mutating x
        $X' := X' \cup \{x'\}$
    end for
    Selection phase:
    Create the multiset $X^{(t+1)}$, the population at time $t+1$, by deleting the λ individuals with lowest f-value in X'
    $t := t+1$
end while

$\mathbb{R}$. We call n the *dimension* of the *size* of the problem and in our runtime analysis we aim at estimating the runtime as a function of n.

1.2.1 The $(\mu+\lambda)$ EA

The $(\mu+\lambda)$ EA is a simple mutation-based elitist evolutionary algorithm. It stores a population of μ individuals which is initialized with random bit strings of length n. In each iteration of the algorithm, we independently generate λ offspring, each by selecting an individual from the parent population uniformly at random and mutating it. The mutation operators considered in this work are discussed in Section 1.3. Then we select μ individuals with the best fitness among the μ parents and λ offspring into the parent population for the next iteration. In case of a tie offspring are selected before parents, and the remaining ties are broken uniformly at random. We note that the results presented in Chapter 2 hold also for any other tie-breaking mechanism.

In Chapter 3 we consider a special case of the $(\mu+\lambda)$ EA, the $(1+1)$ EA, which has both parent and offspring populations of size one.

The pseudocode of the $(\mu+\lambda)$ EA is shown in Algorithm 1.

For the $(\mu+\lambda)$ EA we call every iteration of the outer loop a *generation*. For $t \in \mathbb{N}$, we define P_t as the parent population of the algorithm after generation t.

There is a special case of the $(\mu+\lambda)$ EA, which we call the $(\lambda+\lambda)$ EA with fair parent selection or the $(\lambda \overset{1:1}{+} \lambda)$ EA for short. It has the parent and offspring population of the same size λ and its main difference is that every individual in the population creates exactly one offspring

Algorithm 2 – The $(\lambda \overset{1:1}{+} \lambda)$ EA, maximizing a given function $f : \{0,1\}^n \to \mathbb{R}$, with population size λ.

Initialization:
Create a population of μ individuals by choosing $x^{(i)} \in \{0,1\}^n$, $1 \leq i \leq \mu$ uniformly at random.
Let the multiset $X^{(0)} := \{x^{(1)}, ..., x^{(\lambda)}\}$ be the population at time 0. Let $t := 0$.
Optimization:
while an optimum has not been reached **do**
    $X' := X^{(t)}$
    Mutation phase:
    for $i = 1, \ldots, \lambda$ **do**
        $x :=$ the i-th individual from $X^{(t)}$ (deterministic selection)
        Create x' by mutating x
        $X' := X' \cup \{x'\}$
    end for
    Selection phase:
    Create the multiset $X^{(t+1)}$, the population at time $t+1$, by deleting the λ individuals with lowest f-value in X'
    $t := t + 1$
end while

in each iteration. The pseudocode of the $(\lambda \overset{1:1}{+} \lambda)$ EA is shown in Algorithm 2. This algorithm was previously studied in [13] on ONEMAX, but we also study it on PLATEAU$_k$ function in Chapter 3.

1.2.2 The (μ, λ) EA

The (μ, λ) EA is a non-elitist evolutionary algorithm, which differs from the $(\mu + \lambda)$ EA only in the selection of the individuals to the next generation. It starts with a population that consists of μ random vectors from $\{0,1\}^n$. Then it repeats the following cycle until some stopping criteria is met. The algorithm chooses an individual x from the population uniformly at random and then creates its offspring by mutating x. After obtaining λ offspring it selects the μ best (in terms of fitness) of them into the next population (ties are broken uniformly at random). Due to this non-elitist selection it is necessary that $\mu \leq \lambda$. The pseudo-code of the (μ, λ) EA is shown in Algorithm 3.

1.2.3 The $(1 + (\lambda, \lambda))$ GA

The $(1 + (\lambda, \lambda))$ GA, first presented in [34], has the following working principles. It stores one current individual x, which is initialized with a random bit string. Each iteration of the $(1 + (\lambda, \lambda))$ GA consists of two phases, which are the *mutation* phase and the *crossover* phase.

Algorithm 3 – The (μ, λ) EA maximizing a given function $f : \{0,1\}^n \to \mathbb{R}$, with population size μ and offspring population size $\lambda \geq \mu$.

Initialization:
Create a population of μ individuals by choosing $x^{(i)} \in \{0,1\}^n$, $1 \leq i \leq \mu$ uniformly at random.
Let the multiset $X^{(0)} := \{x^{(1)}, ..., x^{(\mu)}\}$ be the population at time 0. Let $t := 0$.
Optimization:
while an optimum has not been reached **do**
    $X' := \emptyset$ *// the only line different from the* $(\mu + \lambda)$ *EA*
    Mutation phase:
    for $i = 1, \ldots, \lambda$ **do**
        Choose $x \in X^{(t)}$ uniformly at random
        Create x' by mutating x
        $X' := X' \cup \{x'\}$
    end for
    Selection phase:
    Create the multiset $X^{(t+1)}$, the population at time $t+1$, by deleting the $\lambda - \mu$ individuals with lowest f-value in X'
    $t := t + 1$
end while

In the mutation phase the algorithm first chooses the *mutation strength* ℓ following the binomial distribution with parameters n and p, where p is usually called the *mutation rate*. It then creates λ mutants by copying the current individual x and flipping exactly ℓ bits which are chosen uniformly at random, independently for each mutant. After that the mutant with the best fitness is chosen as the winner of the mutation phase x' (all ties are broken uniformly at random). In the crossover phase the algorithm λ times performs a crossover between x and x' by taking each bit from x' with probability c and from x otherwise. The probability c is called the *crossover bias*. The best crossover offspring y (all ties are again broken uniformly at random) is compared with the current individual x. If y is not worse, then it replaces x. The main hope behind this algorithm is that with a high mutation rate, the mutation winner x' contains some beneficial solution elements, and that the crossover with the parent acts as a repair mechanism that removes the destructions caused by the high mutation rate. The runtime analyses for this algorithm on OneMax and on random satisfiability instances (see Section 1.5) confirm that, indeed, the $(1 + (\lambda, \lambda))$ GA shows this behavior on these instances. The pseudocode of the $(1 + (\lambda, \lambda))$ GA optimizing a pseudo-Boolean function f is shown in Algorithm 4.

The standard parameter setting proposed in [34] uses the mutation rate $p = \frac{\lambda}{n}$ and the crossover bias $c = \frac{1}{\lambda}$. These parameters guarantee that if the mutation winner contains some beneficial bit (and differs from the parent by $O(\lambda)$ bits, which is very likely), then with constant probability there is a crossover offspring that has all bits repaired apart from the beneficial one.

Another argument for this parameterization is that a single application of mutation and crossover with the parent, without intermediate selection, would create an offspring distributed as if generated via standard bit mutation with mutation rate $\frac{1}{n}$. Note that $\frac{1}{n}$ is the usual recommendation for the mutation rate in standard bit mutation (though [57] suggests that this is not so clear).

Algorithm 4 – The $(1+(\lambda,\lambda))$ GA maximizing $f\colon \{0,1\}^n \to \mathbb{R}$

$x \leftarrow$ random bit string of length n
while not terminate **do**
Mutation phase:
  Choose $\ell \sim \mathrm{Bin}(n,p)$
  for $i \in [1..\lambda]$ **do**
    $x^{(i)} \leftarrow$ a copy of x
    Flip ℓ bits in $x^{(i)}$ chosen uniformly at random
  end for
  $x' \leftarrow \arg\max_{z \in \{x^{(1)},\dots,x^{(\lambda)}\}} f(z)$
Crossover phase:
  for $i \in [1..\lambda]$ **do**
    Create $y^{(i)}$ by taking each bit from x' with probability c and by taking it from x with probability $(1-c)$
  end for
  $y \leftarrow \arg\max_{z \in \{y^{(1)},\dots,y^{(\lambda)}\}} f(z)$
  if $f(y) \geq f(x)$ **then**
    $x \leftarrow y$
  end if
end while

In our proofs we often exploit the following elementary observation.

Lemma 1. *Let x be some bit string of length n. Let $\mathcal{M}_1$ be the mutation operator*[1] *that first chooses $\ell \sim \mathrm{Bin}(n,p)$ and then flips exactly ℓ random bits. Let $\mathcal{M}_2$ be a standard bit mutation operator that flips each bit independently with probability p. Then for all $y \in \{0,1\}^n$ we have*

$$\Pr[\mathcal{M}_1(x) = y] = \Pr[\mathcal{M}_2(x) = y].$$

This simple observation allows us to consider each particular mutant as being generated via standard bit mutation with mutation rate $\frac{\lambda}{n}$. Note that we cannot do so with the mutation winner x' since its distribution is significantly affected by the selection.

1.3 Mutation and Crossover Operators

In this section we describe the variation operators which are considered in this work and recall some existing theoretical results for them as well as several analysis tools which are specific for these operators. We study only two types of the operators: *unary* operators (or *mutation* operators), which have a single argument and return a single individual, and *binary* operators (or *crossover* operators) which have two arguments and return a single individual. We do not consider operators of a higher-arity and operators which return more than one individual.

[1] The mutation operators are discussed in more details in Section 1.3

1.3.1 Unbiased Operators

All algorithms considered in this work use only unbiased mutation and crossover operators. A mutation operator $\mathcal{M}$ for bit-string representations is called *unbiased* if it is symmetric in the bit-positions $[1..n]$ and in the bit-values 0 and 1. This is equivalent to saying that for all $x \in \{0,1\}^n$ and all automorphisms σ of the hypercube $\{0,1\}^n$ (respecting Hamming neighbors) we have $\sigma^{-1}(\mathcal{M}(\sigma(x)) = \mathcal{M}(x)$, which is an equality of distributions. Similarly, the crossover operator $\mathcal{X}$ is unbiased if it is symmetric in the bit-positions $[1..n]$ and in the bit-values 0 and 1 of both its arguments. Using the same language it is equivalent to say that $\sigma^{-1}(\mathcal{X}(\sigma(x),\sigma(y))) = \mathcal{X}(x,y)$. The notation of unbiasedness was introduced (also for higher-arity operators) in the seminal paper [97].

For our purposes, it suffices to know that the set of unbiased mutation operators consists of all operators which can be described as follows. For the unbiased mutation operators we first choose a number $\ell \in [0..n]$ then we flip exactly ℓ bits chosen uniformly at random. Therefore, the only difference between all unbiased mutation operators is in how they choose ℓ. This characterization of these operators can be derived from [56, Proposition 19]. It was explicitly stated in [39]. Examples for unbiased mutation operators are the operator of Random Local Search, which flips a single random bit, or standard bit mutation, which flips each bit independently with probability $\frac{1}{n}$. Note that in the first case ℓ is always equal to one, whereas in the latter ℓ follows a binomial distribution with parameters n and $\frac{1}{n}$. In the $(1+(\lambda,\lambda))$ GA we explicitly use an unbiased mutation: we first choose ℓ and then flip ℓ randomly chosen bits.

The class of unbiased mutation operators contains a few operators which are unable to solve even very simple problems. For example, operators that always flips exactly two bits never finds the optimum of any function with unique optimum if the initial individual has an odd Hamming distance from the optimum. To avoid such artificial difficulties, we only consider unbiased operators that have at least a constant (positive) probability to flip exactly one bit. In Chapter 3 we also touch the operators with a sub-constant (but still positive) probability to flip exactly one bit.

For the unbiased crossover operators the similar description is more complicated. The reason is that although unbiased operators cannot treat bits differently depending on their values and positions, they do can distinguish bits which are the same in both bit strings participating in the crossover and which are different. Hence, unbiased crossover operator decides how many bit-flips it makes in these two bit sets and then flips the chosen number of bits in each set choosing their positions uniformly at random.

For a more formal description consider a single application of an unbiased crossover operator to x and y. Let L be the set of indices of equal bits in x and y and let M be the set of indices of bits which differ. Any unbiased crossover creates x', which is a copy of one of its arguments. Then it chooses numbers $\ell_L \in [0..|L|]$ and $\ell_M \in [0..|M|]$, chooses uniformly at random ℓ_L bits in positions from L and ℓ_M bits in positions from M and flips the chosen bits in x', which is returned as a result. The for the unbiased crossover used in the $(1+(\lambda,\lambda))$ GA ℓ_L is always zero and ℓ_M follows $\mathrm{Bin}(|M|,\frac{1}{\lambda})$.

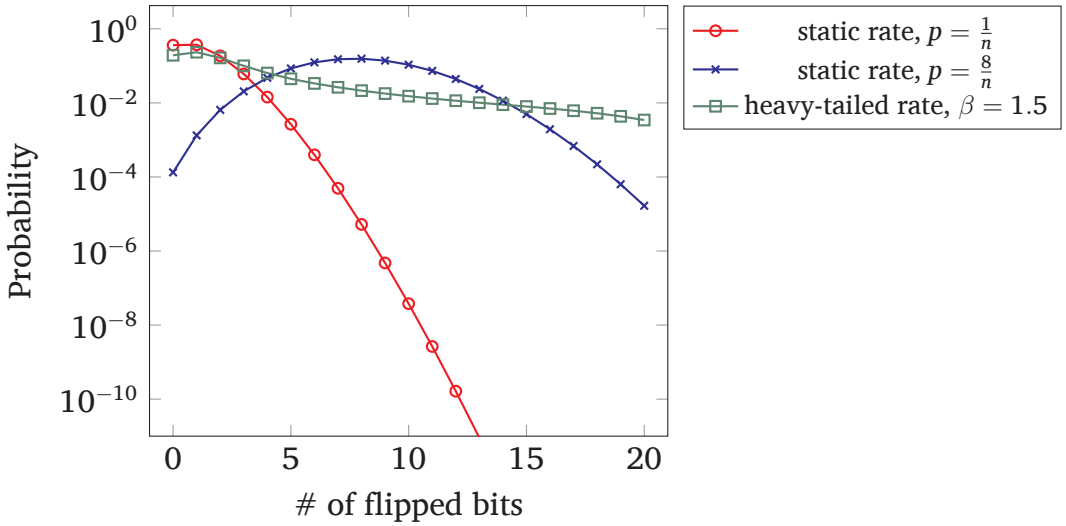


Figure 1 – *The probability to flip ℓ bits for $n = 40$. The heavy-tailed rate gives a more balanced distribution of the number of bits that are flipped.*

1.3.2 Fast Mutation Operator and Power-law Distribution

In Chapter 4 we base our results on the *fast mutation operator*, an unbiased mutation operator which uses the *power-law distribution*. This operator first chooses some integer number α from the power-law distribution and then performs a standard bit mutation with mutation rate $\frac{\alpha}{n}$.

We say that a random variable $X \in \mathbb{N}$ follows a power-law distribution $\mathrm{pow}(\beta, u)$ with parameters β and u if

$$\Pr[X = i] = \begin{cases} C_{\beta,u} i^{-\beta}, & \text{if } i \in [1..u], \\ 0, & \text{else,} \end{cases}$$

where $C_{\beta,u} = (\sum_{j=1}^{u} j^{-\beta})^{-1}$ is the normalization coefficient. We write $X \sim \mathrm{pow}(\beta, u)$ and call u the upper limit of X and β the power-law exponent.

The idea of the fast mutation operator comes from the observation that the number of bits flipped by the standard bit mutation with rate p follows the binomial distribution $\mathrm{Bin}(n, p)$ and therefore it is concentrated around pn. Hence, choosing low p, namely $p = (\frac{1}{n})$, makes it not likely to flip super-constant number of bits, which might lead to a bad performance on multi-modal problems [57]. On the other hand, choosing $p = \omega(\frac{1}{n})$ makes it not likely to flip exactly one bit, which reduces the exploitation efficiency. The fast mutation operator overcomes both issues, since it chooses the mutation rate from the power-law distribution and thus it has a good probability to flip any number of bits (see Figure 1). It was previously shown that this operator can relieve the algorithm user from choosing the optimal mutation rate for a relatively small price. Inspired by this idea, in Chapter 4 we show that this way of random choice of parameters can be even more effective in the $(1 + (\lambda, \lambda))$ GA.

The benefits of the fast mutation operator are based on the following properties of the power-law distribution. Consider some $X \sim \mathrm{pow}(u, \beta)$. Then we have

1) If $\beta > 1$, then $\Pr[X = i] = \Theta(1)$ for any integer $i = \Theta(1)$.

2) The distribution is *heavy-tailed*, which means that we have a decent (only inverse polynomial instead of negative-exponential) probability that $X = i$ for any super-constant $i \leq u$.
3) If $\beta > 2$, then we also have $E[X] = \Theta(1)$.

These properties are easily seen from the following estimates of the partial sums of the generalized harmonic series.

Lemma 2. *For all positive integers a and b such that $b \geq a$ and for all $\beta > 0$, the sum $\sum_{i=a}^{b} i^{-\beta}$ is*
- $\Theta((b+1)^{1-\beta} - a^{1-\beta})$, *if* $\beta \in [0, 1)$,
- $\Theta(\log(\frac{b+1}{a}))$, *if* $\beta = 1$, *and*
- $\Theta(a^{1-\beta} - (b+1)^{1-\beta})$, *if* $\beta > 1$,

where we use Θ notation with respect to $b \to +\infty$.

This lemma follows from the following two lemmas.

Lemma 3. *For all $u \geq 1$ and for all $\beta \neq 1$ we have $\sum_{i=1}^{\lfloor u \rfloor} i^{-\beta} \geq \frac{u^{1-\beta}-1}{1-\beta}$. For $\beta = 1$ we have $\sum_{i=1}^{\lfloor u \rfloor} i^{-\beta} \geq \ln(u)$.*

Proof. We estimate the sum for $\beta \neq 1$ through the corresponding integral.

$$\sum_{i=1}^{\lfloor u \rfloor} i^{-\beta} \geq \int_1^u x^{-\beta} dx = \frac{u^{1-\beta} - 1}{1-\beta}.$$

The case for $\beta = 1$ is a well-known bound on the partial sum of the harmonic series.

□

Lemma 4. *For all $u \in \mathbb{N}$ we have*
- $\sum_{i=1}^{u} i^{-\beta} \leq u^{1-\beta} \frac{2-\beta}{1-\beta}$, *if* $\beta < 0$,
- $\sum_{i=1}^{u} i^{-\beta} \leq \frac{u^{1-\beta}}{1-\beta}$, *if* $\beta \in [0, 1)$,
- $\sum_{i=1}^{u} i^{-\beta} \leq \frac{\beta}{\beta-1}$, *if* $\beta > 1$,
- $\sum_{i=1}^{u} i^{-\beta} \leq \ln(u) + 1$, *if* $\beta = 1$.

Proof of Lemma 4. By analogy with Lemma 3 we estimate the sum through a corresponding integral. If $\beta < 0$ we have

$$\sum_{i=1}^{u} i^{-\beta} \leq \int_1^u x^{-\beta} dx + u^{-\beta} \leq \frac{u^{1-\beta} - 1}{1-\beta} + u^{-\beta} \leq u^{1-\beta} \frac{2-\beta}{1-\beta}.$$

If $\beta \geq 0$ we have

$$\sum_{i=1}^{u} i^{-\beta} \leq 1 + \int_2^{u+1} (x-1)^{-\beta} dx \leq 1 + \frac{u^{1-\beta} - 1}{1-\beta}.$$

If $\beta \in [0, 1)$, then we have

$$\sum_{i=1}^{u} i^{-\beta} \leq \frac{u^{1-\beta} - 1 + 1 - \beta}{1-\beta} \leq \frac{u^{1-\beta}}{1-\beta}.$$

If $\beta > 1$, we have

$$\sum_{i=1}^{u} i^{-\beta} \leq 1 + \frac{1}{\beta - 1} \leq \frac{\beta}{\beta - 1}.$$

The case for $\beta = 1$ is a well-known bound on the partial sum of the harmonic series. □

Lemma 2 gives the following estimates for the normalization coefficient $C_{\beta,u}$ of the power-law distribution and for the expected value of $X \sim \text{pow}(\beta, u)$.

Lemma 5. *The normalization coefficient $C_{\beta,u} = (\sum_{j=1}^{u} i^{-\beta})^{-1}$ of the power-law distribution with parameters β and u is*

- $\Theta(u^{\beta-1})$*, if* $\beta \in [0, 1)$*,*
- $\Theta(1/\log(u+1))$*, if* $\beta = 1$*, and*
- $\Theta(1)$*, if* $\beta > 1$*,*

where we use Θ notation with respect to $u \to +\infty$.

Lemma 6. *The expected value of $X \sim \text{pow}(\beta, u)$ is*

- $\Theta(u)$*, if* $\beta < 1$*,*
- $\Theta(\frac{u}{\log(u)})$*, if* $\beta = 1$*,*
- $\Theta(u^{2-\beta})$*, if* $\beta \in (1, 2)$*,*
- $\Theta(\log(u+1))$*, if* $\beta = 2$*, and*
- $\Theta(1)$*, if* $\beta > 2$*,*

where we use Θ notation with respect to $u \to +\infty$.

1.4 Model Functions

Most theoretical studies analyse EAs on some model functions in order to observe the behavior of an algorithm on a landscape with some particular features. In this work we consider four model functions described in the following subsections after discussing the term of the black-box complexity.

1.4.1 Black-box Complexity

The runtime analysis of evolutionary algorithms is closely connected with the term of black-box complexity. The *black-box complexity* of a problem is the time in which it is possible to solve the problem with an algorithm for which the problem is a black box. This means that the algorithm cannot have an access to the problem structure and all it can do is to make queries to the problem to learn the value of the optimized function at a search point. As well as in the runtime analysis of EAs the time is measured in the number of queries which an algorithm makes

before making a query with the optimal solution. And since by a problem we usually understand a set of instances, we compute this time by the worst-case instance.

More formally, consider the class of algorithms $\mathcal{A}$ and the problem $\mathcal{F}$. Let $T(A,f)$ denote the runtime of an algorithm A on a problem instance f (which is a random variable, since the algorithm A might be stochastic). Then the black-box complexity of the problem $\mathcal{F}$ for the class of algorithms $\mathcal{A}$ is

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

In the context of the theory of evolutionary computation black-box complexity is often regarded in order to show how close the runtime of a considered algorithm is to the best theoretically possible runtime. Studying the black-box complexity of a problem might also help to understand what the existing algorithms are missing and to design new more effective algorithms based on this insight. Probably the best example of such result is the development of the $(1+(\lambda,\lambda))$ GA which was an outcome of a study of the black-box complexity of ONEMAX problem [34].

In this work we consider only the algorithms which use unbiased unary and binary operators (see Section 1.3). Hence, when discussing different functions, we are interested in the following types of their black-box complexity.

1) *Unrestricted black-box complexity* — the black-box complexity with no restrictions on the algorithms.
2) *Unary unbiased black-box complexity* — the black-box complexity for the algorithms which can only use unbiased mutation operators to generate new queries.
3) *Binary unbiased black-box complexity* — the black-box complexity for the algorithms which can only use unbiased mutation and crossover operators operators to generate new queries.

In the rest of this section we discuss the black-box complexity of the considered problems in order to give the reader a better understanding of how hard these problems are for the evolutionary algorithms. For more details on the black-box complexity see survey [62].

1.4.2 ONEMAX

Function ONEMAX : $\{0,1\}^n \to \mathbb{R}$ is defined by $\text{ONEMAX}(x) = \sum_{i=1}^{n} x_i$ for all $x \in \{0,1\}^n$. In other words, ONEMAX returns the number of one-bits in its argument. Without proof we note that due to the unbiasedness of the operators used by all considered algorithms all our results related to ONEMAX also hold for the so-called *generalized* ONEMAX function, denoted by ONEMAX_z. This function has some hidden bit-string z and returns the number of coinciding bits in its argument and z. In other words,

$$\text{ONEMAX}_z(x) = \sum_{i=1}^{n}(1-|z_i - x_i|) = n - H(x,z),$$

where $H(x,z)$ stands for the Hamming distance.

The main features of the ONEMAX landscape is that it is a strictly monotone function (every bit flipped from zero to one increases the value) and that it has an ideal fitness-distance

correlation. Analyzing EAs on ONEMAX one can observe how good these EAs on easy stages of the optimization (those where we have a clear signal towards the right bit positions from the fitness) and how good these EAs are at exploitation (whether they meet the coupon collector effect).

Despite its simplicity, ONEMAX has given a birth to many fundamental results, e.g. [64, 65, 66, 87, 42, 139, 121, 6, 33].

The unary black-box complexity of ONEMAX (and also of the generalized ONEMAX) is $\Theta(n\log(n))$ [97, Theorem 6]. The binary unbiased black-box complexity has not been determined precisely yet. What is known is that it is at least $\Omega(\frac{n}{\log(n)})$ (which is the unrestricted black-box complexity of the generalized ONEMAX proven in [68]) and is at most $O(n)$, which is the runtime of the $(1+(\lambda,\lambda))$ GA with optimal fitness-dependent parameters [33].

1.4.3 LEADINGONES

The LEADINGONES function returns the maximal length of a prefix that consists only of ones. More formally, the LEADINGONES function is defined by

$$\textsc{LeadingOnes}(x) = \sum_{i=1}^{n}\prod_{j=1}^{i} x_j$$

for all $x \in \{0,1\}^n$.

This function has a much weaker fitness-distance correlation than ONEMAX: even the search point in distance 1 from the optimum can have the minimal fitness. Hence, the analysis of an algorithm on LEADINGONES shows how much the algorithm relies on this correlation. However, it is a monotone function, so it is considered to be relatively easy for the EAs.

The unary unbiased black-box complexity of LEADINGONES is $\Theta(n^2)$ (the lower bound is shown in [97] and the upper bound is shown in [123] for the $(1+1)$ EA). For the crossover-based algorithms the black-box complexity of LEADINGONES has not been determined precisely, but it is known that it is $\Omega(n\log\log(n))$ [97] and $O(n\log(n))$ [58].

1.4.4 Jump Functions

The $\textsc{Jump}_k$ function with parameter $k \in [2..n]$, which is the width of the valley of low fitness, is then defined as follows.

$$\textsc{Jump}_k(x) = \begin{cases} \textsc{OM}(x)+k, & \text{if } \textsc{OM}(x) \in [0..n-k]\cup\{n\},\\ n-\textsc{OM}(x), & \text{if } \textsc{OM}(x) \in [n-k+1..n-1]. \end{cases}$$

A plot of $\textsc{Jump}_k$ is shown in Figure 2. The fitness is essentially the fitness of ONEMAX except for all search points with Hamming distance between one and $k-1$ from the optimum.

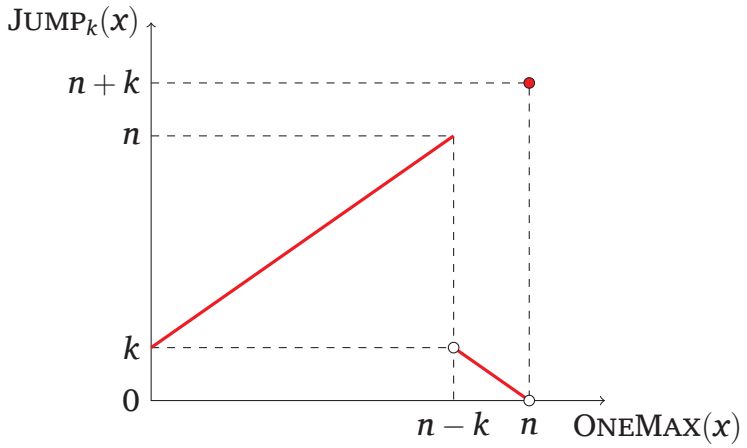


Figure 2 – *Plot of the JUMP$_k$ function. As a function of unitation, the function value of a search point x depends only on the number of one-bits in x*

The unbiased black-box complexity (both unary and binary) of JUMP$_k$ is the same as the one of ONEMAX. This is because if we know the value of JUMP$_k$ at some search point, we can also compute its ONEMAX value and vice versa.[2] While this implies that there is a unary unbiased black-box algorithm finding the optimum of JUMP$_k$ in $O(n \log n)$ time, such results generally do not indicate that a problem is easy for reasonable evolutionary algorithms. For example, in [35] it was shown that the NP-complete partition problem also has a unary unbiased black-box complexity of $O(n \log n)$.

1.4.5 Plateau Functions

In this work we also propose a new benchmark function—PLATEAU$_k$. The PLATEAU$_k$ function resembles the ONEMAX function, but has a plateau of second-highest fitness of radius k around the optimum.

$$\text{PLATEAU}_k(x) := \begin{cases} \text{ONEMAX}(x), & \text{if } \text{ONEMAX}(x) \leq n-k, \\ n-k, & \text{if } n-k < \text{ONEMAX}(x) < n, \\ n, & \text{if } \text{ONEMAX}(x) = n. \end{cases}$$

The plateau of the function PLATEAU$_k(x)$ consists of all bit-strings that have at least $n-k$ one-bits, except the optimal bit-string $x^* = (1, \ldots, 1)$. See Fig. 3 for an illustration of PLATEAU$_k$. Our hope is that this generic fitness function with a plateau of scalable size may aid the understanding of plateaus in evolutionary computation in a similar manner as the jump functions have led to many useful results about the optimization of functions with true local optima, e.g., [67, 90, 36, 12, 19, 20, 71, 16, 14, 57, 20, 134, 79, 23, 25].

[2] To avoid such tricks a slightly different definition of JUMP$_k$ is used in the context of the black-box complexity [36], but even this hard variant of the problem has an unbiased black-box complexity which is much lower than the runtime of reasonable EAs on it.

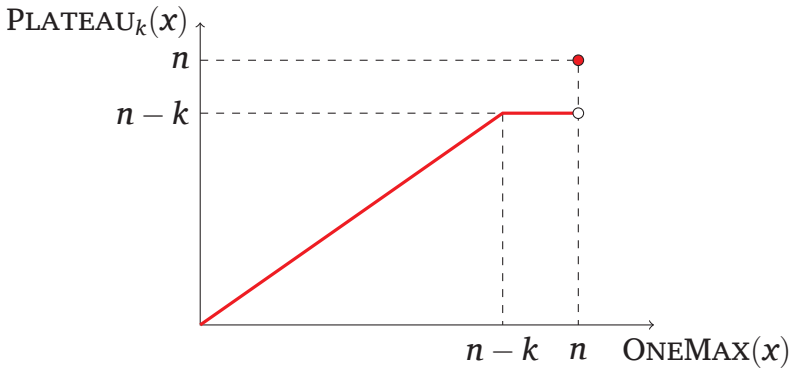


Figure 3 – *Plot of the PLATEAU function. As a function of unitation, the function value of a search point x depends only on the number of one-bits in x*

In this work we consider only PLATEAU$_k$ functions with a constant parameter k. The unary unbiased black-box complexity of these functions is $\Theta(n \log n)$. Since we are first to introduce this benchmark function, we prove its unary black-box complexity in the following Lemma.

Lemma 7. *For all constants k, the unary unbiased black-box complexity of the PLATEAU$_k$ function is $\Theta(n \log n)$.*

Proof. The lower bound follows from the $\Omega(n \log n)$ lower bound for the unary unbiased black-box complexity of ONEMAX shown in [97]. Since we can write PLATEAU$_k = f \circ$ ONEMAX for a suitable function f (such that $f(x) = x$, if $x \notin [n-k..n]$ and $f(x) = n-k$ otherwise), any algorithm solving PLATEAU$_k$ can be transferred into an algorithm which treats all points with fitness in $[n-k..n-1]$ as points with fitness $(n-k)$ and therefore solving ONEMAX in the same time.

The upper bound follows along the same lines as the $O(n \log n)$ upper bound for the unary unbiased black-box complexity of JUMP$_k$, see [36] and note that the algorithm given there contains a sub-routine which, in expected constant time, for a given constant radius r determines the Hamming distance $H(x, x^*)$ of a point x from the optimum x^* without evaluating search points y with $H(y, x^*) \leq r$. Note that the Hamming distance from the optimum determines the ONEMAX value of x. Hence with this routine one can optimize both jump and plateau functions by simulating an $O(n \log n)$ black-box algorithm for ONEMAX. □

1.5 State of the art

In this section we show what is already known about how the algorithms described in Section 1.2 perform on functions described in Section 1.4 and discuss what is missing in this field.

1.5.1 Results for the $(\mu+\lambda)$ EA on OneMax

Before this work there were no known tight bounds on the runtime of the $(\mu+\lambda)$ EA on OneMax. However, the tight bounds were obtained for its special cases—the $(1+\lambda)$ EA and the $(\mu+1)$ EA which have trivial parent and offspring population respectively. In [49], the runtime of the $(1+\lambda)$ EA on the class of linear functions is analyzed, which contains the OneMax function. A tight bound of $\Theta(\frac{n\log n}{\lambda}+\frac{n\log^+\log^+\lambda}{\log^+\lambda})$ is proven for the expected runtime (number of iterations until the optimum is found) of the $(1+\lambda)$ EA maximizing the OneMax function. This extends the earlier result [87], which shows this bound for $\lambda = O(\frac{\log(n)\log\log(n)}{\log\log\log(n)})$ (note that in this case the bound simplifies to $\Theta(\frac{n\log(n)}{\lambda})$) and which shows further that for asymptotically larger values of λ, the expected runtime is $\omega(\frac{n\log(n)}{\lambda})$.

Witt [138] studied the $(\mu+1)$ EA on the three pseudo-Boolean functions LeadingOnes, OneMax and SPC. For the OneMax problem, under the mild assumption that μ is polynomially bounded in n, he proved that the expected runtime of the $(\mu+1)$ EA is $\Theta(\mu n + n\log n)$.

For algorithms with non-trivial parent and offspring population sizes, the following is known. The only work regarding the classic $(\mu+\lambda)$ EA for general μ and λ is [116]. Using the recent switch analysis technique [140] and assuming that μ and λ are polynomially bounded in n, it was shown that the $(\mu+\lambda)$ EA needs an expected number of

$$\Omega\left(\frac{n\log n}{\lambda}+\frac{\mu}{\lambda}+\frac{n\log\log n}{\log n}\right) \tag{1}$$

iterations to find the optimum of any function $f:\{0,1\}^n\to\mathbb{R}$ with unique optimum. Unfortunately, this bound was not accompanied by any upper bound.

For the $(\lambda \overset{1:1}{+} \lambda)$ EA the first result [80, Theorem 4] considers the runtime on the OneMax in a special case when $\lambda = n$. It shows an upper bound of $O(n)$ iterations, which is tight as shown in Section 2.2.4. For the $(\lambda \overset{1:1}{+} \lambda)$ EA with general λ, Chen et al. [13, Proposition 4] show an optimization time of $O(\frac{n\log n}{\lambda}+n\log\lambda)$ iterations. They conjecture a runtime of $O(\frac{n\log n}{\lambda}+n\log\log n)$ [13, Conjecture 3], which is asymptotically at least as good and which is stronger for $\lambda=\omega(\log n)$.

We also find notable that the runtime bound for the (μ,λ) EA noted in Subsection 1.5.2 can be seen as the upper bound for the $(\mu+\lambda)$ EA, since the population of the $(\mu+\lambda)$ EA is always better than the population of (μ,λ) EA after the same number of iterations in the dominating sense.

From these results one can see that we are still missing the asymptotically precise estimate for the runtime of the $(\mu+\lambda)$ EA on OneMax. The results from [49] and [138] for the special cases have not been extended for a general case with non-trivial population sizes for the following reasons. For the $(1+\lambda)$ EA it turned to be non-trivial to extend the results on the $(\mu+\lambda)$ EA due to much more complicated population dynamics compared to the dynamics of a single individual. The methods used for the analysis of the $(\mu+1)$ EA such as family trees (see Section 1.6) become too complicated when more than one offspring is created in each iteration. All mentioned results for the $(\mu+\lambda)$ EA with non-trivial populations (both parent and offspring) are either for some special cases of the $(\mu+\lambda)$ EA, or they are not precise enough as we show in Subsection 2.2.2 (and therefore, they do not give the right insights about the algorithm behavior). Hence, to

obtain the precise bounds on the runtime and to understand the interaction between the parent and offspring population sizes we need to develop some new analysis methods.

1.5.2 Results for the (μ, λ) EA on ONEMAX

What is known about how the (μ, λ) EA optimizes ONEMAX is roughly the following. When the offspring population size λ is at most $(1-\varepsilon)e\mu$ for some positive constant ε, then the expected runtime (measured by the number of fitness evaluations until an optimum is found) is exponential in n [96]. When $\lambda \geq (1+\varepsilon)e\mu$ and $\lambda \geq C \ln n$ with C a sufficiently large constant, then the runtime becomes polynomial, and in fact, $O(n\lambda)$ [17].

There is a good reason for these results. Let x be a parent individual with high fitness, that is, ONEMAX(x) is close to n and thus $d := d(x) := n - \text{ONEMAX}(x)$ is small. When generating offspring from x via standard-bit mutation with mutation rate $\frac{1}{n}$, then with probability roughly $\frac{1}{e}$ the offspring has the same fitness as the parent, with probability $\Theta(\frac{d}{n})$ the offspring is better than the parent, and else it is worse. Consequently, when d is small, the number of individuals with best fitness in the population, in expectation, increases per iteration by a factor of at least $(1+\varepsilon)$ when $\lambda \geq (1+\varepsilon)e\mu$ and it decreases by a factor of roughly $(1-\varepsilon)$ when $\lambda \leq (1-\varepsilon)e\mu$. In the efficient case, the $(1+\varepsilon)$ multiplicative increase of the number of top-individuals suffices to ensure that a single top individual has a constant chance to take over the whole population in $O(\log \mu)$ iterations. In the inefficient case, the size of the subpopulation of the best individuals decreases by the factor of $(1-\varepsilon)$ in each iteration, which leads to its loss much earlier than a superior offspring is generated. We note that a number of highly non-trivial arguments [96, 17] are necessary to transform these observations into rigorous proofs for the runtimes cited above.

While these results are mathematically non-trivial despite their intuitive explanations, they only discuss the easy situations where the number of top individuals is a subject to a clear drift, either into the right or the wrong direction. These situations might be too extreme to lead to a full understanding of the population dynamics of this EA. Moreover, these are typically the situations in which using comma selection is not a good idea. For the case of negative, but also positive drift the main advantage of comma selection is absent. We recall that comma selection is used, among others, with the hope that by not keeping good parent individuals in the population, one can prevent premature convergence. If $\lambda \leq (1-\varepsilon)e\mu$ or λ is too small, then the algorithm does not converge at all (not only prematurely) and is not likely to approach the optimum to a reasonably close distance in polynomial time. If $\lambda \geq (1+\varepsilon)e\mu$ and λ is sufficiently large, then discarding the parent population does not help, since with high probability it reappears in the offspring population.

To be more precise, let us assume that we have a parent population that is converged to a local optimum. Then with $\lambda \geq (1+\varepsilon)e\mu$, an expected number of $(1+\varepsilon)\mu$ copies of this parent are generated as offspring. Since these are generated independently, with probability $1 - e^{-\Omega(\mu)}$ (which follows from Chrenoff bounds, see Lemma 22 in Section 1.8) at least μ such copies are generated, which means that inferior offspring cannot enter the population. For this reason, the two regimes with clear drift are possibly not the most interesting ones for using an EA with comma selection.

The very general analyses of non-elitist EAs in [96, 95, 18, 17] which give as special case the results for the (μ, λ) EA has a downside that it gives the non-expert less understanding

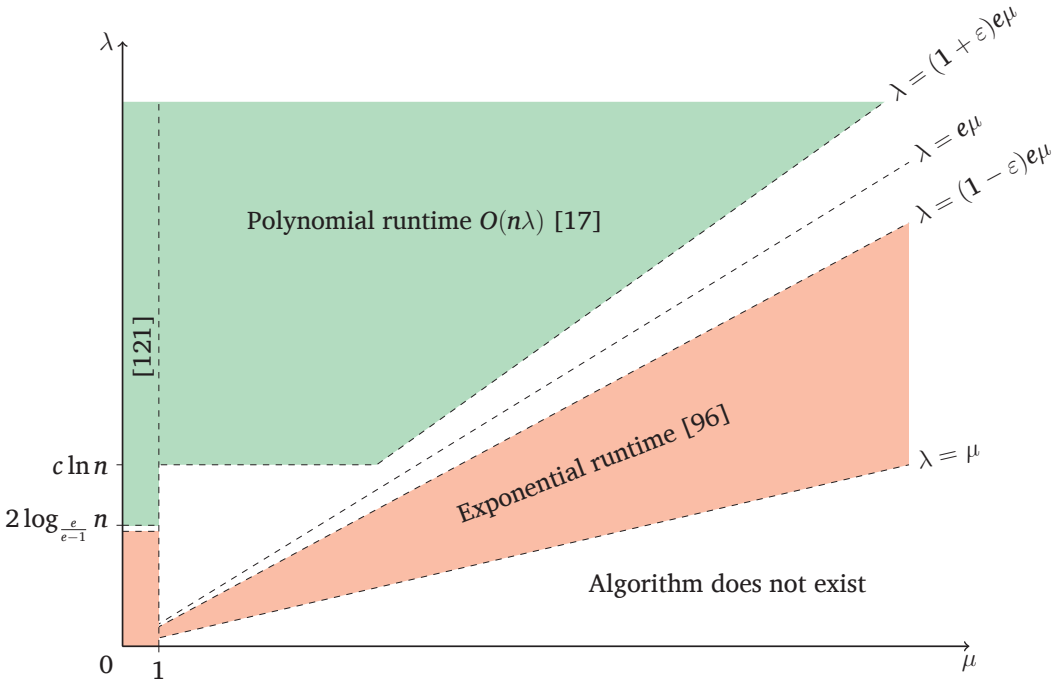


Figure 4 – *The theoretically proven results for the runtime of the* (μ, λ) *EA in its parameters space.*

of how the (μ, λ) EA really solves a problem. This is particularly true for the general results for upper bounds [95, 18, 17], which are proven via an intricate potential function argument. Consequently, our insight delivered in Subsection 2.3.3 that in a run of the (μ, λ) EA with $\lambda \geq (1+\varepsilon)e\mu$, the best individual with constant probability takes over the whole population in $O(\log \mu)$ iterations is not easily derived from these works.

For the special case of the (μ, λ) EA with trivial parent population called the $(1, \lambda)$ EA a threshold behavior was observed in [84, 109, 121]. The latest of these works [121] shows that for $\lambda \geq \log_{\frac{e}{e-1}} n \approx 2.18 \ln n$ the $(1, \lambda)$ EA optimizes ONEMAX in an expected number of $O(n \log n + \lambda n)$ fitness evaluations, whereas for $\lambda \leq (1-\varepsilon) \log_{\frac{e}{e-1}} n$ with constant ε, the runtime is $\exp(\Omega(n^{\varepsilon/2}))$ with high probability. See Figure 4 for an illustration of the existing results for the (μ, λ) EA.

1.5.3 Results for the $(1 + (\lambda, \lambda))$ GA on ONEMAX

While most classic evolutionary algorithms optimize the ONEMAX function in expected time $\Theta(n \log n)$ or slower, it was shown in [33] that the $(1 + (\lambda, \lambda))$ GA does so in expected time

$$O\left(\max\left\{\frac{n \log n}{\lambda}, \frac{n\lambda \log\log \lambda}{\log \lambda}\right\}\right),$$

an expression that is asymptotically smaller than $\Omega(n \log n)$ (the runtime of such algorithms as the $(1+1)$ EA [108], the $(\mu+1)$ EA [138] and the $(1+\lambda)$ EA [87, 49]) for $\lambda \leq \log n$ (and a

slightly larger range) and that gives a bound of slightly better than $O(n\sqrt{\log n})$ when using the optimal value of λ, which is close to $\sqrt{\log n}$ [31].

Using the fitness-dependent parameter choice $\lambda = \sqrt{\frac{n}{n-f(x)}}$, we can achieve a $\Theta(n)$ runtime [33]. For the static and even more the fitness-dependent setting, one has to question if the typical algorithm user would have found good parameter settings. For this reason, approaches that do not require manually finding a good static or fitness-dependent parameter settings appear preferable.

One such approach in which the value of λ is controlled according to a simple one-fifth rule was proposed in [31]. It was proven that this modification lets the $(1 + (\lambda, \lambda))$ GA find the optimum of the ONEMAX function in $\Theta(n)$ fitness evaluations, which is asymptotically the same as when using the optimal fitness-dependent value of λ. However, it was shown in [11] that for a more real-world problem, namely for the random satisfiability instances, the dynamic choice of the parameter λ must be modified to give the good results. More precisely, the maximal value of λ must be capped with $2\ln(n+1)$. This method of parameter control also leads to a very inefficient performance on jump functions [126]. These raises the need to find new universal methods of the efficient parameter control for the $(1 + (\lambda, \lambda))$ GA.

1.5.4 Results for the LEADINGONES

The classical LEADINGONES benchmark function was first proposed in [123]. This function is still easy in the sense that it is unimodal, that is, from every search point there is a path to the optimum such that each edge on this path refers to a one-bit flip increasing the fitness. However, the fitness-distance correlation is low, since all bits to the right of the left-most zero have no influence on the fitness.

We are aware of the following runtime results for the LEADINGONES. The $(1 + \lambda)$ EA, for any polynomial value of λ, finds the optimum of LEADINGONES in an expected number of $\Theta(n^2/\lambda + n)$ iterations, as was shown in [87]. Regarding other mutation-based EAs, it is worth mentioning that $(1 + 1)$ EA needs $\Theta(n^2)$ iterations to find the optimum, as it was shown in [67], and $(\mu + 1)$ EA needs $\Theta(n^2 + \mu n \log n)$ iteration, as it was shown in [138]. For the more detailed overview on the LEADINGONES problem see [48] (slightly more detailed in the arXiv version), which considers some algorithms with sub-quadratic runtime as well. Although these few algorithms are interesting for a further investigation, they are still far from being a generally accepted evolutionary algorithm.

1.5.5 Results for Plateaus

When trying to analyze how evolutionary algorithms optimize plateau functions, we observe that the active area of theoretical analyses of evolutionary algorithms has produced many strong tools suitable to analyze how evolutionary algorithms make true progress (e.g., various form of the fitness level method [133, 125, 18, 17] or drift analysis [81, 44, 98, 47]), but much less is known on how to analyze plateaus.

This is not to mean that plateaus have not been analyzed previously, see, e.g., [72, 88, 8, 43, 110, 69, 70], but these results appear to be more ad hoc and less suitable to derive generic methods for the analysis of plateaus. In particular, with the exception of [72], we are not aware of any results that determine the runtime of an evolutionary algorithm on a fitness function with non-trivial plateaus precise including the leading constant (whereas a decent number of very precise results have recently appeared for unimodal fitness functions, e.g., [7, 41, 139, 102, 83, 37, 82]).

The design of the $\textsc{Plateau}_k$ function is partially inspired by the XDIVK function, which was studied mostly in the context of problems with auxiliary objectives [10, 3]. This function is defined as

$$\textsc{XdivK}(x) := \lfloor \textsc{OneMax}(x)/k \rfloor,$$

where k is an integer parameter. The XDIVK function, which has multiple plateaus, was introduced to study the behavior of the random search heuristics in the absence of the strong signal towards the optimum from the fitness, however the theoretical analysis even of the most simple EAs on this function turned to be a non-trivial problem. In some sense, $\textsc{Plateau}_k$ is a simplified version of XDIVK which lets us observe algorithms' behavior in the hardest plateau of XDIVK and to develop new analysis methods for plateaus.

Another problem with plateaus which received an attention from theoretical community is the ROYALROAD function. This function also has a parameter k and is defined[3] as

$$\textsc{RoyalRoad}(x) := \sum_{i=0}^{n/k-1} \prod_{j=1}^{k} x_{ik+j}.$$

In simple words, we divide a bit string into blocks of length k and each block adds k to the fitness if and only if all its bits are equal to one. This functions family was introduced in [106] to study the building blocks hypothesis, but later it was also used to study the effect of crossover [124, 89] and population sizes [50] on plateaus.

1.5.6 Results for Jump Functions

Clearly the usual application of evolutionary algorithms are problems with multimodal landscapes, that is, with non-trivial local optima, and these local optima usually present a difficulty for evolutionary algorithms. In the runtime analysis perspective multimodal problems have displayed very different optimization behaviors. For example, on multimodal landscapes it has been observed that crossover can recombine solutions into significantly better ones [90], that mutation rates significantly larger than $\frac{1}{n}$ can be preferable [57], and that estimation-of-distribution algorithms can significantly outperform classic algorithms [79, 23].

When we consider the theoretical studies of jump functions we note that for the elitist algorithms the only way to leave the local optimum of $\textsc{Jump}_k$ to a strictly better search point is to flip exactly the right k bits and go to the optimum. For this reason, it comes as no surprise that many mutation-based evolutionary algorithms need $\Omega(n^k)$ time to optimize such a jump

[3] We only consider the definition for the case when k is a delimiter of the problem size n in order to simplify notation

function. Using the right mutation rate [57] or the stagnation detection mechanism [118] can reduce this time down to $\Omega((\frac{n}{k})^k)$. Non-elitist comma selection although designed to avoid getting stuck in local optima does not show a better performance [29]. Crossover can be helpful, but the maybe most convincing work [20] in this direction also only obtains a runtime of $O(n^{k-1}\log n)$ with the standard mutation rate and $O(n^{k-1})$ with a higher mutation rate. With additional tools, runtimes up to $O(n)$ were obtained [21, 71, 134], but the lower the runtimes become, the more these algorithms are custom-tailored to jump functions (e.g., they intensively use the symmetry of JUMP functions, a feature which is not often met in practice). The extreme end is marked by an $O(\frac{n}{\log n})$ time algorithm [12] designed in the context of unrestricted black-box complexity.

1.5.7 Summary for the Current State of the Art

Regarding the existing results one can notice the following gaps in the theoretical knowledge about EAs.

1) We do not know the true runtime and the behavior of the $(\mu+\lambda)$ EA on ONEMAX. This is especially dissatisfying in the light of that we already have precise results for the special cases, the $(1+\lambda)$ EA and the $(\mu+1)$ EA, but the key understanding of the interplay of the two non-trivial population sizes is missing.
2) So far the behavior of the (μ,λ) EA on ONEMAX has been studied theoretically only for the parameter settings which do not exploit the main idea behind the comma selection, which is, to be able to leave local optima while preserving the ability to converge to the global optimum. This might lead us to a wrong conclusion that the comma selection should not be used at all, while the threshold parameters are promising to be able to hold the balance between the elitist behavior and the premature convergence.
3) It is not well-understood how the populations help to leave local optima and traverse through plateaus. There are some results which show that the crossover-based algorithms with diversity mechanisms cope with local optima on the example of JUMP function, but they exploit the symmetry of JUMP function and do not have much practical support of their efficiency.
4) The best-known performance of a crossover-based algorithm on ONEMAX is obtained via the dynamic parameter choices in the $(1+(\lambda,\lambda))$ GA. However, both known ways to choose the parameters (fitness-dependent and according to the one-fifth rule) do not work well on more real-world problems like MAX-3SAT. Therefore, to increase the efficiency of the $(1+(\lambda,\lambda))$ GA on practical problems we need to find new universal ways of dynamic parameter choices.

The reason why these parts of the knowledge about EAs is still uncovered by theoretical studies is that the existing tools are not able to deliver such results. In the following section we overview the theoretical toolbox and explain why these tools cannot be used to answer the stated questions.

1.6 Existing Tools

The spectrum of mathematical tools used for the runtime analysis is extremely wide and includes the means from probability theory, linear algebra, mathematical analysis and other disciplines. However, there is a group of tools which are most frequently used in the theory of evolutionary computation and it is fair to say that they form a basis for all theoreticians in this field. In this section we make an overview of these tools.

1.6.1 Markov Chains

Markov chains are a widely used tool for the runtime analysis of evolutionary algorithms (see, e.g., [107, 130, 122]). In this work we only regard *absorbing* Markov chains. A Markov chain is called absorbing if there is a subset S' of the set of its states S such that

(1) for every state $s_1 \in S$ there exists a state $s_2 \in S'$ such that there exists a path of transitions with positive probabilities from s_1 to s_2 (we call s_2 an *absorbing* state) and
(2) for every absorbing state $s \in S'$ the probability to leave this state is zero.

Absorbing chains appear naturally in runtime analysis. When taking as states of the Markov chain the possible states of the algorithm, we can assume the optima to be absorbing. The runtime of the algorithm is the number of transitions in the chain until it reaches an absorbing state. We only regard absorbing Markov chains with exactly one absorbing state.

The standard way to compute the expected number of steps until an absorbing state reaching uses the *fundamental matrix*, which is built as follows. Let P be the transition matrix of an absorbing Markov chain, that is, the matrix where each element p_i^j is equal to the transition probability from state i to state j. Let Q be the square submatrix of P consisting only of the rows and columns which correspond to transient states of the chain. We call Q the *transient matrix* for brevity. Then the fundamental matrix N of this chain is defined as

$$N = \sum_{t=0}^{+\infty} Q^t = (I - Q)^{-1},$$

where I is the identity matrix of the same order as Q. Let π be a stochastic vector which represents the initial distribution over the transient states. Then the expected time until we reach an absorbing state is

$$E[T] = \pi N \mathbb{1} = \|\pi N\|_1 ,$$

where $\mathbb{1}$ is a column vector of all ones.

However, working with fundamental matrix is not convenient, since it might be hard to compute its elements precisely. Alternatively, we can use matrix Q to compute the runtime as

$$E[T] = \sum_{t=0}^{+\infty} \|\pi Q^t\|_1 . \tag{2}$$

Markov chains are frequently used for the runtime analysis of EAs, however only when they describe a relatively simple process, that is, either a simple algorithm with trivial population

or only a component of a more complicated process. As soon as we introduce populations, the set of states of the considered algorithm expands significantly, and the structure of the Markov chain which describes the run of the algorithm becomes too complicated for the analysis.

1.6.2 Drift Analysis

Drift analysis is one of the most efficient tools for the runtime analysis of the random search heuristics. It includes a set of theorems, which help to estimate the expected runtime or even the tail bounds via the expected progress of the considered process. All drift theorems originate from the negative drift theorem by Hajek [78]. Here we show its interpretation from [113] which is easier to understand and apply than the original theorem.

Theorem 1 (negative drift theorem)**.** *Let X_t, $t = 0, 1, 2, \dots$ be real-valued random variables describing a stochastic process over some state space. Let $a, b \in \mathbb{R}$ such that $d := b - a > 0$. Let T be the first point in time when $X_t \geq b$ conditional on that $X_0 \leq a$. If there exist $\gamma > 0$ and $p > 0$ such that*

$$E[e^{\gamma(X_{t+1}-X_t)} \mid X_t \in (a, b)] \leq 1 - \frac{1}{p}$$

holds for all t, then for any time bound L we have

$$\Pr[T < L] \leq LDpe^{-\gamma d},$$

where $D := \max_{t\in\mathbb{N}}\{1, E[e^{\gamma(X_{t+1}-a)} \mid X_t \leq a]\}$.

The big step forward the drift analysis was the additive drift theorem by He and Yao [80].

Theorem 2 (additive drift theorem)**.** *Let $\{X_t\}_{t\in\mathbb{N}}$ be a sequence of random variables that describe some random process over a finite state space $S \subset [a, b]$ such that $b \in S$. Let T be the first time when $X_t = b$.*

1) *If there exist some δ such that for all $s \in S \setminus \{b\}$ and $t \in \mathbb{N}$ we have $E[X_{t+1} - X_t \mid X_t = s] \leq \delta$, then $E[T] \geq \frac{b-E[X_0]}{\delta}$.*
2) *If there exist some δ such that for all $s \in S \setminus \{b\}$ and $t \in \mathbb{N}$ we have $E[X_{t+1} - X_t \mid X_t = s] \geq \delta$, then $E[T] \leq \frac{b-E[X_0]}{\delta}$.*

We note that this theorem can also estimate the time until the random process X_t reaches its lowest state. For this it is sufficient to apply this theorem to another process $Y_t = -X_t$.

The additive theorem has given a raise to a series of other drift theorems: multiplicative drift [45], variable drift [98] and others. The most recent overview of the existing techniques can be found in [100].

To perform drift analysis of an algorithm one has to define a potential function over all possible states of the considered algorithm. The drift theorems are then applied to this potential function. Despite the diversity and effectiveness of drift theorems, it is a challenging task to use them for the analysis of the population-based algorithms, since they require to design a potential function which takes into account the whole population. Also sometimes they are not capable to deliver tail bounds on the runtime or they are used on a high level, so that valuable insights are obscured.

1.6.3 Fitness Levels

Fitness levels theorems [96, 17] are mostly designed for the non-elitist algorithms except for the artificial fitness levels [135]. They cover a wide range of processes, but the price for this universality is that they give only little insight on the process they are applied to. This is because these techniques use only some properties of the considered process which lie on the surface (and this is what grants them the universality) but they do not use some deeper machinery of the process. E.g., different fitness levels theorems were used to prove the runtime of the (μ, λ) EA on ONEMAX when there is a clear drift in the number of the best individuals in the population but they did not meet the conditions to be applied to the threshold parameter values (see Subsection 1.5.2).

The artificial fitness levels theorem [135] is used to deliver the upper bounds on the runtime of elitist algorithms. In this work we use its following interpretation (which is a simplified version of the original one, hence we also include a simple proof).

Theorem 3. *Let the space S of all possible populations of some population-based algorithm be divided into m disjoint sets $A_1, \dots, A_m$ that are called* levels*. We write $A_{\geq i} = \bigcup_{j=i}^{m} A_j$ for all $i \in [1..m]$.*

Let P_t be the population of the algorithm after iteration t. Assume that for all $t \geq 0$ and $i \in [2..m]$, we have that $P_t \in A_i$ implies $\Pr[P_{t+1} \in A_{\geq i}] = 1$. Let T be the minimum number t such that $P_t \in A_m$.

1) *Assume that there are $T_1, \dots, T_{m-1} \geq 0$ such that for all $t \geq 0$ and $i \in [1..m-1]$ we have that if $P_t \in A_i$, then $E[\min\{s \mid s \in \mathbb{N}, P_{t+s} \in A_{\geq i+1}\}] \leq T_i$ (for all possible $P_0, \dots, P_{t-1}$). Then*
$$E[T] \leq \sum_{i=1}^{m-1} T_i.$$
2) *Assume that there are $p_1, p_2, \dots, p_{m-1}$ such that for all $t \geq 0$ and $i \in [1..m-1]$ we have that if $P_t \in A_i$, then $\Pr[P_{t+1} \in A_{\geq i+1}] \geq p_i$ (for all possible $P_0, \dots, P_{t-1}$). Then*
$$E[T] \leq \sum_{i=1}^{m-1} \frac{1}{p_i}.$$

Proof. We start by proving the first claim. Consider a run of the algorithm. Let $t_i = \min\{t \mid P_t \in A_{\geq i}\}$. Consider some $i \in [1..m-1]$. We analyze the random variable $t_{i+1} - t_i$. If there is no t with $P_t \in A_i$, then $t_i = t_{i+1}$ simply by the definition of the t_i. Otherwise, by our assumptions, we have $E[t_{i+1} - t_i] \leq T_i$. Note that this applies trivially also to the first case where we just saw that $t_{i+1} - t_i = 0$. Hence, from $T = t_m = \sum_{i=1}^{m-1}(t_{i+1} - t_i)$ we conclude
$$E[T] = \sum_{i=1}^{m-1} E[t_{i+1} - t_i] \leq \sum_{i=1}^{m-1} T_i.$$
To prove the second claim, we note that by our assumptions $t_{i+1} - t_i$ is stochastically dominated by a geometric distribution with success rate p_i. Hence $E[t_{i+1} - t_i] \leq \frac{1}{p_i}$, and the claim follows as above. □

Theorem 3 is an efficient tool to deliver the upper bounds. Even for the processes which can have a super-constant expected progress it is often possible to divide the search space into

levels in such way that this theorem gives asymptotically precise upper bounds on the runtime (we do it in Subsection 2.2.1). However, to deliver the lower bounds with similar arguments one usually needs to make additional assumptions about the processes (e.g., in [127, 135]), which does not let us make a universal version of this theorem for the lower bounds.

In this work when using the fitness levels technique, we use the following language. We say that *the algorithm is on level i* if the current population is in the level A_i. We also say that *the algorithm gains a level* or *the algorithm leaves the current level* if the new population is at the higher level than the previous one.

1.6.4 Family Trees

The main idea of Witt's [137, 138, 136] family tree argument is to consider a tree graph the vertices of which are the individuals created during the evolution process and each path from the root to a vertex corresponds to the series of mutations which led to the creation of this vertex. Selection does not play any role in this structure, so when working with family trees we usually assume that all individuals with a corresponding vertex in the tree can potentially be present in the current population.

To use the family tree argument one should first argue that with high probability the true family tree has a certain structure (e.g., a small height) and then, conditioning on this, argue that within such a restricted structure an optimal solution is hard to reach as it was done in the following lemma in [138].

Lemma 8 (Lemma 2 in [138]). *Let $D(t)$ denote the depth of a family tree of the $(\mu+1)$ EA at time t. For all $t \geq 0$ we have $\Pr[D(t) \geq 3t/\mu] = 2^{\omega(t/\mu)}$.*

Although family trees give us an interesting view on the evolution process an let us disregard the selection, so far they have been applied only to the $(\mu+1)$ EA. When non-trivial offspring populations come into scene, the tree structure changes and some properties (including the one from Lemma 8) do not hold with probability which is high enough. Therefore, the domain where this tool can be applied is too narrow.

1.6.5 Summary of the Existing Methods and Motivation of This Work

In this section we have shown that although the theory of evolutionary computations developed a strong collection of runtime analysis tools, we still do not have sufficient means to analyse population-based EAs and understand the true role of populations. The main problems we face are the following ones.

- The population-based EAs have too many states, which makes it hard to design an easy-to-analyse Markov's chain or a potential function for the drift analysis.
- The application of the fitness levels theorems usually does not give enough understanding of the algorithm behavior.

- The family trees are designed for the algorithms with trivial offspring populations, which makes it hard to use them for the analysis of the $(\mu + \lambda)$ EA.

The populations are believed to be essential for the EAs and are commonly used by practitioners. The insufficient theoretical understanding of how the populations affect the runtime of EAs makes it hard to use them efficiently in practice. E.g., it might be a non-trivial task to find an optimal population size. The main theoretical goal of this work is to develop new analysis tools which would let us analyse population-based EAs and distill valuable practical recommendations from analyses performed with these tools. The main practical goal of this work is to obtain some valuable insights about the nature of populations in EAs using the developed toolbox and based on these insights give some recommendations on how to set population sizes. Our interest is not only at finding optimal static values of this parameter, but also at developing new efficient strategies of the dynamic choice of population size.

1.7 Contribution of This Work

The main contribution of this work to the field of evolutionary computation consists of three parts.

Analysis methods. We propose the following new analysis methods for the population-based algorithms.

- *Complete trees technique* which lets us derive lower bounds on runtime of a population-based EA in Subsection 2.2.2.
- *Method of analysis of no-drift processes* which lets us analyse processes with no drift via the drift analysis. This turns out to be useful in the analysis of population dynamics in Subsection 2.3.2.
- *Method of analysis of EAs on plateaus*.
- Additive drift theorem *with tail bounds* which can give strong bounds on the runtime concentration for the processes which have a constant drift.

New algorithm, called the *heavy-tailed* $(1 + (\lambda, \lambda))$ *GA*, which is based on the $(1 + (\lambda, \lambda))$ GA and uses the power-law distribution for the dynamic parameters choice, described in Section 4.1.

Analyses of the population-based EAs. Using the developed methods we show the following results.

- We show the asymptotically tight runtime of the $(\mu + \lambda)$ EA on ONEMAX function.
- We show the surprising influence of the absolute population size on the performance of the (μ, λ) EA solving ONEMAX when it uses threshold parameters.
- We show that the choice of the mutation operator almost does not have an impact on the performance of the mutation-based EAs on plateaus.
- We perform an analysis of the proposed heavy-tailed $(1 + (\lambda, \lambda))$ GA on ONEMAX, LEADINGONES and JUMP_k. This analysis shows its effectiveness and universality.

1.8 Useful Tools

In this section we collect the tools which are used in our analyses. These tools in contrast with the ones mentioned in Section 1.6 are not specialized on the delivery of runtime bounds, but they are frequently used in the theory of evolutionary computation and help to prove some essential properties of analysed processes.

We start with the union bound which is an elementary, but useful result from probability theory which helps us give an upper bound on the probability of a union of several events.

Lemma 9 (union bound)**.** *Let A and B be some events in some probability space. Then*

$$\Pr[A \cup B] \leq \Pr[A] + \Pr[B].$$

We show the following two common estimates for the distribution of the fitness of an individual y which is created via the standard bit mutation to an individual x when we optimize the ONEMAX function.

Lemma 10. *Let f denote the ONEMAX function. Let x be an individual of fitness $f(x) = n - d$. Let y be the result of the standard bit mutation with mutation rate $\frac{1}{n}$ applied to x. Then, for all $i \geq 1$, we have*

$$\Pr[f(y) - f(x) = i] \leq \binom{d}{i} \left(\frac{1}{n}\right)^i.$$

Proof. To improve fitness exactly by i we need to flip at least i zero-bits. The probability that we flip at least i bits is $\frac{1}{n^i}$. By the union bound over all possible choices of i zero-bits which we flip, we prove the lemma. □

Lemma 11. *Let f denote the ONEMAX function. Let x be an individual of fitness $f(x) = n - d$. Let y be the result of the standard bit mutation with mutation rate $\frac{1}{n}$ applied to x. Then*

$$\Pr[f(y) - f(x) = 0] = \sum_{k=0}^{\min\{d,n-d\}} \binom{d}{k}\binom{n-d}{k}\left(\frac{1}{n}\right)^{2k}\left(1-\frac{1}{n}\right)^{n-2k}.$$

This probability can be bounded with

$$(1-\frac{1}{n})^n \leq \Pr[f(y) - f(x) = 0] \leq \frac{1}{e} \cdot \frac{1}{1-\frac{d(n-d)}{(n-1)^2}}.$$

Proof. The probability to preserve fitness is the probability that we flip the same number of zero-bits and one-bits. For all $i \in [0..\min\{d, n-d\}]$ the probability to flip exactly i zero-bits (and not to flip other $d-i$ zero-bits) is $\binom{d}{i}(\frac{1}{n})^i(1-\frac{1}{n})^{d-i}$ and the probability to flip the same number of one-bits is $\binom{n-d}{i}(\frac{1}{n})^i(1-\frac{1}{n})^{n-d-i}$. Summing up the probability of these two independent events over $i \in [0..\min\{d, n-d\}]$ we get the expression for $\Pr[f(y) - f(x) = 0]$. Note also that if we flip more than $\min\{d, n-d\}$ bits with the same value, the fitness will be different.

After that, the bounds on $\Pr[f(y) - f(x) = 0]$ are obtained as follows. The lower bound $(1-\frac{1}{n})^n$ is just the first term of the sum. The upper bound $\frac{1}{1-\frac{d(n-d)}{(n-1)^2}}$ follows from the observation that the terms of the sum are bounded by the geometric progression with ratio $\frac{d(n-d)}{(n-1)^2}$. □

For two real random variables X and Y we say that *Y stochastically dominates X* if for all $k \in \mathbb{R}$ we have $\Pr[Y \geq k] \geq \Pr[X \geq k]$. See [27] for a more detailed description of this concept. In that case, we use the notation $X \preceq Y$. We use this notion to argue and make precise that better parents generate better offspring. The following result is from [139].

Lemma 12. *Let $x, y \in \{0,1\}^n$ such that $f(x) \leq f(y)$. Let $X = \mathcal{M}(x)$ and $Y = \mathcal{M}(y)$, where $\mathcal{M}$ is an unbiased mutation operator. Then $f(X) \preceq f(Y)$.*

We also use the following fact.

Lemma 13. *Let X and Y be two random variables over $\mathbb{N}$ such that $X \preceq Y$. Then $E[X] \leq E[Y]$.*

A *coupling* for two random variables X and Y is a pair of random variables $(\tilde{X}, \tilde{Y})$ defined over the same probability space such that X and $\tilde{X}$ as well as Y and $\tilde{Y}$ follow the same law. The following result is well-known.

Theorem 4. *Let X and Y be two random variables. Then the two following statements are equivalents.*

1) $X \preceq Y$.

2) There exists a coupling $(\tilde{X}, \tilde{Y})$ such that $\tilde{X} \leq \tilde{Y}$.

In our proofs we use the following elementary estimates (Lemmas 14—19).

Lemma 14. *For all $x \geq 0$ and all $a > 0$ we have $xe^{-\frac{x}{a}} \leq \frac{a}{e}$.*

Proof. Let $a > 0$. Consider the function $g_a(x) = xe^{-\frac{x}{a}}$. Notice that $g_a(0) = 0$ and $\lim_{x \to +\infty} g_a(x) = 0$. In the interval $[0, +\infty)$ function $g_a(x)$ is positive and smooth (i.e., with continuous derivative), hence its maximum can be only in the roots of its derivative $g'_a(x)$. We compute

$$g'_a(x) = e^{-\frac{x}{a}} - \frac{x}{a}e^{-\frac{x}{a}}.$$

Note that $g'_a(x) = 0$ only when $x = a$. The value of g_a at this point is $ae^{-\frac{a}{a}} = \frac{a}{e}$. Therefore, $g_a(x) \leq \frac{a}{e}$ for all $x \geq 0$ and $a > 0$. □

Lemma 15. *For all $x \geq 0$ and all $a > 0$ we have $x^2 e^{-\frac{x}{a}} \leq \frac{4a^2}{e^2}$*

The proof repeats the one of Lemma 14.

Lemma 16. *For all $x > 0$ we have $e^{-2x}(1+x) \leq \max\{1 - \frac{x}{2}, 1 - \frac{1}{4}\}$.*

Proof. Consider $g(x) \coloneqq e^{-2x}(1+x)$. We first compute its first and second derivatives.

$$g'(x) = e^{-2x}(1 - 2(1+x)) = e^{-2x}(-2x - 1),$$
$$g''(x) = e^{-2x}(-2 - 2(-2x - 1)) = 4xe^{-4x}.$$

For all $x \geq 0$ $g'(x)$ is negative and $g''(x)$ is positive, which means that $g(x)$ is decreasing and convex in $[0, +\infty)$. Therefore, for all $x \in [0, \frac{1}{2}]$ we have

$$g(x) \leq g(0) + x\frac{g\left(\frac{1}{2}\right) - g(0)}{\frac{1}{2} - 0} = 1 - \left(2 - \frac{3}{e}\right)x \leq 1 - \frac{x}{2}.$$

For all $x > \frac{1}{2}$ we have

$$g(x) \le g\left(\frac{1}{2}\right) \le 1 - \frac{1}{4}.$$

Combining these two observations for two intervals we prove that $g(x) \le \max\{1 - \frac{x}{2}, 1 - \frac{1}{4}\}$. □

Lemma 17. *For all $x \in [0,1]$ we have $e^x(1 - 2(e-1)x) \le 1 - (e-2)x$*

Proof. Since e^x is convex downwards, for all $x \in [0,1]$ we have $e^x \le 1 + ex$. Hence,

$$e^x(1 - 2(e-1)x) \le (1 + ex)(1 - 2(e-1)x) \le 1 - (e-2)x.$$

□

Lemma 18. *For all $x \ge -\delta > -1$ we have $\ln(1+x) \ge x - \frac{x^2}{2} - \frac{\delta^3}{3(1-\delta)^3}$.*

Proof. Applying the Taylor's theorem with the remainder in the Lagrange form to $\ln(1+x)$ in point $x_0 = 0$ we obtain

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3(1+c)^3},$$

where c is some value between x and zero. Since by the lemma condition $x \ge -\delta$, we have

$$\ln(1+x) \ge x - \frac{x^2}{2} - \frac{\delta^3}{3(1-\delta)^3}.$$

□

Lemma 19. *Let $d = o(1)$. Then we have $(1 - \frac{1}{n})^d = (1 - \frac{d}{n} - o(\frac{d}{n}))$.*
We also have $(1 - \frac{1}{n})^n = \frac{1}{e}(1 - O(\frac{1}{n}))$.

Proof. Using the Taylor's formula for the natural logarithm and for the exponential function we obtain

$$\begin{aligned}\left(1 - \frac{1}{n}\right)^d &= \exp\left(d \ln\left(1 - \frac{1}{n}\right)\right) = \exp\left(d\left(-\frac{1}{n} - o\left(\frac{1}{n}\right)\right)\right) \\ &= \exp\left(-\frac{d}{n} - o\left(\frac{d}{n}\right)\right) = \left(1 - \frac{d}{n} - o\left(\frac{d}{n}\right)\right).\end{aligned}$$

With the same arguments we have

$$\begin{aligned}\left(1 - \frac{1}{n}\right)^n &= \exp\left(n \ln\left(1 - \frac{1}{n}\right)\right) = \exp\left(n\left(-\frac{1}{n} - O\left(\frac{1}{n^2}\right)\right)\right) \\ &= \exp\left(-1 - O\left(\frac{1}{n}\right)\right) = \frac{1}{e}\exp\left(-O\left(\frac{1}{n}\right)\right) \\ &= \frac{1}{e}\left(1 - O\left(\frac{1}{n}\right)\right).\end{aligned}$$

□

To be able to make the transition between the number of iterations and the number of fitness evaluations in algorithms with a variable population size, we use Wald's equation [132].

Lemma 20 (Wald's equation)**.** *Let $(X_t)_{t\in\mathbb{N}}$ be a sequence of real-valued random variables and let T be a positive integer random variable. Let also all following conditions be true.*

1) All X_n have the same finite expectation.

2) For all $t \in \mathbb{N}$ we have $E[X_t \mathbb{1}_{\{T\geq t\}}] = E[X_t] \Pr[T \geq t]$.

3) $\sum_{t=1}^{+\infty} E[|X_t| \mathbb{1}_{\{T\geq t\}}] < \infty$.

4) $E[T]$ is finite.

Then we have

$$E\left[\sum_{t=1}^{T} X_t\right] = E[T]E[X_1].$$

We use the following inequality to estimate the probability that at least one of λ Bernoulli trials succeeds.

Lemma 21. *For all $p \in [0,1]$ and all $\lambda > 0$ we have*

$$1-(1-p)^\lambda \geq \frac{\lambda p}{1+\lambda p} \geq \frac{1}{2}\min\{1,\lambda x\}.$$

Proof. By [121, Lemma 8] we have $(1-x)^\lambda \leq \frac{1}{1+\lambda x}$. Hence,

$$\begin{aligned} 1-(1-x)^\lambda &\geq 1-\frac{1}{1+\lambda x} = \frac{\lambda x}{1+\lambda x} \\ &\geq \frac{\lambda x}{2\max\{1,\lambda x\}} = \frac{1}{2}\min\{\lambda x, 1\}. \end{aligned}$$

□

Random variable X follows binomial distribution $\mathrm{Bin}(n,p)$ with parameters n and p if it is a sum of n independent random variables which follow Bernoulli distribution with parameter p. The binomial distribution appears naturally in the theory of evolutionary computation. For example, the number of bits flipped by the standard bit mutation or the number of copies of the best individuals in the population both follow binomial distributions (but with different parameters). We now collect several tools to simplify our calculations when we work with this distribution.

We start with Chernoff bounds (see Theorems 1.10.1 and 10.10.5 in [30]) which we use to show the concentration of some random variables involved in our analysis. We use the following lemma, which is a particular case of these bounds for the random variables following a binomial distribution.

Lemma 22 (Chernoff Bounds)**.** *Let X be a random variable following a binomial distribution $\mathrm{Bin}(n,p)$. Then for all $\delta \in (0,1)$ the probability that $X \geq (1+\delta)np$ is at most $e^{-\frac{\delta^2 np}{3}}$ and the probability that $X \leq (1-\delta)np$ is at most $e^{-\frac{\delta^2 np}{2}}$. Also for all $\delta \geq 1$ the probability that $X \geq (1+\delta)np$ is at most $e^{-\frac{\delta np}{3}}$.*

In our proofs we shall use the following result for random variables with binomial distribution from [77]. An elementary proof for it was given in [24, Theorem 10].

Lemma 23. *Let $X \sim \mathrm{Bin}(n,p)$ such that $p > 1/n$. Then $\Pr(X \geq E[X]) > 1/4$.*

The next lemma follows from the Chernoff bounds and shows the concentration of the number of the bit flips in the mutation phase of the $(1+(\lambda,\lambda))$ GA by the Chernoff bounds.

Lemma 24. *Let $p \geq \frac{1}{n}$. Then the number ℓ of the bits flipped by the mutation operator of the $(1+(\lambda,\lambda))$ GA is in $[pn, 2pn]$ with at least constant probability $q_\ell \geq 0.1$, if n is at least some sufficiently large constant.*

Proof. Recall that the number ℓ of the flipped bits in the mutation phase of the $(1+(\lambda,\lambda))$ GA is chosen according to the binomial distribution $\mathrm{Bin}(n,p)$. We first consider the case when p is small. Assume $pn \in [1,9]$. Then

$$\begin{aligned}\Pr\left[\ell \in [pn, 2pn]\right] &\geq \Pr\left[\ell = \lceil pn \rceil\right] = \binom{n}{\lceil pn \rceil} p^{\lceil pn \rceil} (1-p)^{n-\lceil pn \rceil} \\ &\geq \frac{(n-\lceil pn \rceil)^{\lceil pn \rceil}}{n^{\lceil pn \rceil}} \cdot \frac{(pn)^{\lceil pn \rceil}}{\lceil pn \rceil !} \cdot (1-p)^{\frac{1}{p} pn} \geq 0.1.\end{aligned}$$

The last inequality holds if n is large enough (we omit the details, since it is too tedious).

Now we consider the case when $pn \geq 9$. Since ℓ follows the binomial distribution with parameters n and p, we have $E[\ell] = pn$. By the Chernoff bounds we have

$$\Pr\left[\ell \geq 2E\left[\ell\right]\right] \leq \exp\left(-\frac{E\left[\ell\right]}{3}\right) = \exp\left(-\frac{pn}{3}\right),$$

By Lemma 23 we have

$$\Pr[\ell \geq E[\ell] = pn] \geq \frac{1}{4}.$$

Hence,

$$\Pr[\ell < pn] \leq \frac{3}{4}.$$

Therefore, by the union bound the probability q_ℓ that $\ell \in [pn, 2pn]$ is at least

$$\begin{aligned}q_\ell &= \Pr\left[\ell \geq pn \cap \ell \leq 2pn\right] \\ &\geq 1 - \Pr\left[\ell < pn\right] - \Pr\left[\ell > 2pn\right] \\ &\geq 1 - \frac{3}{4} - \exp\left(-\frac{pn}{3}\right).\end{aligned}$$

Since we assume that $pn \geq 9$, we obtain

$$\exp\left(-\frac{pn}{3}\right) \leq \exp\left(-3\right) \leq 0.05$$

and hence,

$$q_\ell \geq 1 - \frac{3}{4} - \exp\left(-\frac{pn}{3}\right) \geq 0.2.$$

Therefore

$$q_\ell \geq 0.1 = \Omega\left(1\right).$$

□

We also state a similar lemma for the larger mutation rates (which are of a greater interest when we aim at escaping a local optimum with the $(1+(\lambda,\lambda))$ GA).

Lemma 25. *Assume $p = \omega(\frac{1}{n})$. Then the number ℓ of the bits flipped by the mutation operator is in $[\frac{1}{2}pn, \frac{3}{2}pn]$ with probability $q'_\ell = 1 - o(1)$.*

Proof. By the Chernoff bounds we have

$$\begin{aligned} q'_\ell &\geq 1 - \Pr\left[\ell < \tfrac{1}{2}pn\right] - \Pr\left[\ell > \tfrac{3}{2}pn\right] \\ &\geq 1 - e^{-\frac{pn}{8}} - e^{-\frac{pn}{12}} = 1 - e^{-\omega(1)} - e^{-\omega(1)} = 1 - o(1). \end{aligned}$$ □

We also encounter random variables with hypergeometric distribution. A particular example of such random variable is the number ℓ_0 of zero-bits which are flipped by the mutation operator of the $(1+(\lambda,\lambda))$ GA after the total number ℓ of the bits to flip is already chosen. This random variable follows a hypergeometric distribution with parameters n, $n - \mathrm{OM}(x)$ and ℓ. For this random variable the Chernoff bounds are also applicable [30, Theorem 1.10.25].

Lemma 26. *Let ℓ_0 be the number of zero-bits of a bit string x such that $OM(x) > \frac{9}{16}n$ which are flipped by the mutation operator of the $(1+(\lambda,\lambda))$ GA after ℓ is chosen. Then the probability that $\ell_0 > \frac{\ell}{2}$ is at most $\exp(-\frac{\ell}{336})$.*

The next tool is the classic result for for martingales. Recall that a *martingale* with respect to the filtration $\mathcal{F}$ is a stochastic process M such that, for all $n \in \mathbb{N}$, we have $E[M_{n+1} \mid \mathcal{F}_n] = M_n$.

Theorem 5 (Doob's Decomposition [63])**.** *For any integrable process $(X_n)_\mathbb{N}$, there exists a martingale $(M_n)_\mathbb{N}$ and a predictable integrable process $(A_n)_\mathbb{N}$ such that $A_0 = M_0 = 0$ and, for all $n \in \mathbb{N}$, $X_n = X_0 + M_n + A_n$. This decomposition is almost surely unique.*

With this theorem we prove the following auxiliary lemma.

Lemma 27. *Let $\lambda \geq e\mu$. Let $X_t \sim \min\{\mu, \mathrm{Bin}(\lambda, \frac{X_{t-1}}{e\mu})\}$. Then for all $t \in \mathbb{N}$ and all $\Delta > 0$, the probability that there exist $\tau \in [1..t]$ such that $X_\tau < X_0 - \Delta$ is at most $\frac{tX_0}{\Delta^2}$.*

Proof of Lemma 27. We define the process M_τ as follows. $M_0 = X_0$ and, for $\tau \geq 0$ and $M_{\tau+1} \sim \mathrm{Bin}\left(\lambda, \frac{M_\tau}{\lambda}\right)$ if $M_\tau > X_0 - \Delta$ and $M_{\tau+1} = M_\tau$ otherwise. By Doob's decomposition theorem (Theorem 5), there exists a martingale $(N_\tau)_\mathbb{N}$ and a predictable process $(A_\tau)_\mathbb{N}$ such that, for all $\tau \in \mathbb{N}$, $(M_\tau)^2 = (M_0)^2 + N_\tau + A_\tau$. Note that M is a martingale. Consequently, for all $\tau \geq 0$, we have

$$\begin{aligned} E[(M_{\tau+1} - M_\tau)^2 \mid \mathcal{F}_\tau] &= E[(M_{\tau+1})^2 - 2M_{\tau+1}M_\tau + (M_\tau)^2 \mid \mathcal{F}_\tau] \\ &= E[(M_{\tau+1})^2 - (M_\tau)^2 \mid \mathcal{F}_\tau] = A_{\tau+1} - A_\tau, \end{aligned}$$

where $\mathcal{F}_\tau$ stands for the natural filtration. We sum these equalities to obtain

$$\sum_{k=0}^{\tau-1} E[(M_{k+1} - M_k)^2 \mid \mathcal{F}_k] = A_\tau = (M_\tau)^2 - (M_0)^2 - N_\tau.$$

Now as N is a martingale with $N_0 = 0$ we have $E[N_\tau] = E[N_0] = 0$, therefore

$$E[(M_\tau)^2] = E[(M_0)^2] + \sum_{k=0}^{\tau-1} E[(M_{k+1} - M_k)^2].$$

Finally, M is a martingale so $E[M_\tau]^2 = E[M_0]^2 = E[(M_0)^2]$, since X_0 is determined. Thus we have

$$\begin{aligned}\mathrm{Var}(M_\tau) &\le \sum_{k=0}^{\tau-1} E[(M_{k+1}-M_k)^2] = \sum_{k=0}^{\tau-1} E[E[(M_{k+1}-M_k)^2 \mid \mathcal{F}_k]] \\ &= \sum_{k=0}^{\tau-1} E[\mathrm{Var}[M_{k+1} \mid \mathcal{F}_k]] \le \sum_{k=0}^{\tau-1} E\left[M_k\left(1-\frac{M_k}{\lambda}\right)\right] \\ &\le \sum_{k=0}^{\tau-1} E[M_k] = \tau X_0.\end{aligned}$$

By Chebyshev's inequality,

$$\Pr[M_\tau \le X_0 - \Delta \mid X_0 = k] \le \frac{\tau k}{\Delta^2}.$$

Therefore as long as $M_\tau \in \{X_0-\Delta+1, \cdots, \mu\}$ we have $M_\tau \preceq X_\tau$. Let $\tau_1 = \inf\{t \in \mathbb{N} \mid M_t > \mu\}$. By Theorem 4, there exists a coupling $(\tilde{X}, \tilde{M})$ such that for all $\tau < \tau_1$, we have $\tilde{M}_\tau \le \tilde{X}_\tau$. Therefore if $\tilde{M}_\tau$ exceeds μ we can wait until $\tilde{X}_\tau \le X_0$ and restart the argument with $t = \tau_1$ and $\tau' = \tau - \tau_1 \le \tau$. Finally we have

$$\Pr[\exists t \in [0..\tau] : X_t \le X_0 - \Delta \mid X_0 = k] \le \frac{\tau k}{\Delta^2}.$$

□

We make a use of the following property of the random variable which is a transformation of another random variable with a binomial distribution.

Lemma 28. *There exists a constant S_{min} such that if $X \sim \mathrm{Bin}(n,p)$ with $np \ge S_{min}$ and $p \le \frac{1}{2}$, then we have*

$$E[\ln(1+X)] \ge \ln(1+np) - \frac{5}{6}\frac{(1-p)}{np}.$$

Proof. We first transform the random variable in the follwoing way.

$$\begin{aligned}E[\ln(1+X)] &= E[\ln(1+np) + \ln(1+X) - \ln(1+np)] \\ &= \ln(1+np) + E\left[\ln\left(1+\frac{X-np}{1+np}\right)\right].\end{aligned}$$

Let $Y = \frac{X-np}{1+np}$, which is a random variable. Note that the $E[Y] = 0$. Then we have

$$E[\ln(1+X)] = \ln(1+np) + E[\ln(1+Y)],$$

hence our aim is to prove that $E[\ln(1+Y)] \ge -\frac{5}{6}\frac{(1-p)}{(1+np)}$.

Consider some $\delta \in (0,1)$, the precise value of which we will choose later. Let D be the event $(X \ge (1-\delta)np)$. Event D also implies that $Y \ge -\frac{\delta np}{1+np} \ge -\delta$. We have

$$E[\ln(1+Y)] = E[\mathbb{1}_D \ln(1+Y)] + E[(1-\mathbb{1}_D)\ln(1+Y)].$$

When D is satisfied, by Lemma 18 we have

$$E[\mathbb{1}_D \ln(1+Y)] \ge E\left[\mathbb{1}_D\left(Y - \frac{Y^2}{2} - \frac{\delta^3}{3(1-\delta)^3}\right)\right].$$

Since event complementary to D includes only negative values of Y, we have

$$E[\mathbb{1}_D \ln(1+Y)] \geq E[Y] - \frac{1}{2}E[Y^2] - \frac{\delta^3}{3(1-\delta)^3} = -\frac{np(1-p)}{2(1+np)^2} - \frac{\delta^3}{3(1-\delta)^3}$$
$$\geq -\frac{(1-p)}{2(1+np)} - \frac{\delta^3}{3(1-\delta)^3},$$

where we used $E[Y] = 0$ and $E[Y^2] = \mathrm{Var}(Y) = \frac{1}{(1+np)^2}\mathrm{Var}(X)$. Let

$$\delta := \frac{\left(\frac{1-p}{2(1+np)}\right)^{\frac{1}{3}}}{1+\left(\frac{1-p}{2(1+np)}\right)^{\frac{1}{3}}} \geq \frac{1}{2}\left(\frac{1-p}{2(1+np)}\right)^{\frac{1}{3}},$$

so that $\frac{\delta^3}{(1-\delta)^3} = (\frac{1-p}{2(1+np)})^{\frac{1}{3}}$. Then

$$E[\mathbb{1}_D \ln(1+Y)] \geq -\frac{2(1-p)}{3(1+np)}$$

By Chernoff bounds, we have

$$\Pr(X \leq (1-\delta)np) \leq \exp\left(-\frac{\delta^2}{2}np\right) \leq \exp\left(-\frac{(1-p)^{2/3}np}{8(1+np)^{2/3}}\right)$$
$$\leq \exp\left(-\frac{1}{8}(1-p)^{2/3}np^{1/3}\right)$$

Since $x \mapsto \ln(1+x)$ is monotonically increasing, we have

$$E[(1-\mathbb{1}_D)\ln(1+Y)] \geq \ln\left(1-\frac{np}{1+np}\right)\exp\left(-\frac{1}{8}(1-p)^{2/3}np^{1/3}\right)$$
$$= -\ln(1+np)\exp\left(-\frac{1}{8}(1-p)^{2/3}np^{1/3}\right)$$

The logarithm slowly grows in np, but the exponent tends to zero much faster (recall that by the lemma conditions we have $p \leq \frac{1}{2}$, therefore $\frac{1}{2}(1-p)^{2/3}$ is a constant). Hence, there exists a constant S_{min} such that if $np \geq S_{min}$, we have

$$E[(1-\mathbb{1}_D)\ln(1+Y)] \geq -\frac{1-p}{6(1+np)}.$$

Therefore,

$$E[\ln(1+Y)] \geq -\frac{2}{3}\frac{(1-p)}{(1+np)} - \frac{1-p}{6(1+np)} = -\frac{5}{6}\frac{(1-p)}{(1+np)}.$$

□

We also make a use of the following adaptation of the additive drift theorem for a random variable that dominates the binomial distribution with a capped value.

Lemma 29. *Consider $\lambda \in \mathbb{N}$ and $\mu \in \mathbb{N}$ such that $\lambda \geq e\mu$. Let X_t and Δ_t be some random processes such that for all $t \in \mathbb{N}$ we have $\Delta_t \geq \Delta_{\min} > 0$ for some $\Delta_{\min} \in (0, \lambda)$ and $X_{t+1} \sim \min\{\mu, \mathrm{Bin}(\lambda, \frac{X_t+\Delta_t}{e\mu})\}$. Let $T(X')$ be the first moment in time when $X_T \geq X'$ for some X' that is at least $\max\{18 \ln \frac{2\lambda}{\Delta_{\min}}, 48\}$, but not greater than $\frac{\mu}{2}$. Then we have*

$$E[T(X')] \leq \max\left\{24, \frac{4X' - 2X_0}{\Delta_{\min}}\right\}.$$

Proof of Lemma 29. Although the drift of X_t towards X' is at least Δ_t, we cannot apply the additive drift theorem from the box, since this drift partially comes from the fact that X_t is surely larger than X', and the additive drift theorem prohibits to jump over the target value.

To overcome this problem we define the potential function $\Phi(X)$ for all $X \in \mathbb{N}$ as follows.

$$\Phi(X) = \begin{cases} 0, \text{ if } X \geq X', \\ 2X' - X, \text{ else.} \end{cases}$$

To ease the notation we introduce another random process $\tilde{X}_t \sim \mathrm{Bin}(\lambda, \frac{X_t+\Delta_t}{e\mu})$. Note that $E[\tilde{X}_t] = X_{t-1} + \Delta_{t-1}$. We also define $p_t(i) \coloneqq \Pr[X_t = i \mid X_{t-1}]$ and $q_t(i) \coloneqq \Pr[\tilde{X}_t = i \mid X_{t-1}]$. Note that if $i < \mu$, then $p_t(i) = q_t(i)$ and $p_t(\mu) = \sum_{i=\mu}^{\lambda} q_t(i)$. Hence, we estimate the expected difference in the potential function after one step of the process as follows.

$$\begin{aligned}
E[\Phi(X_t) - \Phi(X_{t+1}) \mid X_t] &= \sum_{i=0}^{X'-1} p_{t+1}(i)(i - X_t) \\
&\quad + \sum_{i=X'}^{\mu} p_{t+1}(i)(2X' - X_t) \\
&\geq \sum_{i=0}^{2X'} q_{t+1}(i) i + \sum_{i=2X'+1}^{\lambda} q_{t+1}(i) 2X' - X_t \\
&\geq \sum_{i=0}^{\lambda} q_{t+1}(i) i + \sum_{i=2X'+1}^{\lambda} q_{t+1}(i)(2X' - i) - X_t \\
&\geq E[\tilde{X}_{t+1}] - X_t - \sum_{i=2X'+1}^{\lambda} q_{t+1}(i) i \\
&\geq \Delta_t - \lambda \Pr[\tilde{X}_{t+1} > 2X'].
\end{aligned}$$

If $\Delta_t \leq \frac{X'}{2}$, then by Chernoff bounds we have

$$\begin{aligned}
\Pr[\tilde{X}_{t+1} \geq 2X'] &= \Pr\left[\tilde{X}_{t+1} \geq \left(1 + \frac{2X'}{X_t + \Delta_t} - 1\right)(X_t + \Delta_t)\right] \\
&\leq \exp\left(-\left(\frac{2X'}{X_t + \Delta_t} - 1\right)^2 \frac{X_t + \Delta_t}{3}\right) \\
&\leq \exp\left(-\frac{(X' - \Delta)^2}{3(X' + \Delta)}\right) \leq \exp\left(-\frac{X'}{18}\right).
\end{aligned}$$

Since by the lemma conditions we have $X' \geq 18 \ln \frac{2\lambda}{\Delta_{\min}}$, we have $\Pr[\tilde{X}_{t+1} > 2X'] \leq \frac{\Delta_{\min}}{2\lambda}$. Hence we obtain

$$E[\Phi(X_t) - \Phi(X_{t+1}) \mid X_t] \geq \Delta_t - \lambda \frac{\Delta_{\min}}{2\lambda} \geq \frac{1}{2}\Delta_{\min}.$$

Otherwise, if $\Delta_t \geq \frac{X'}{2} > 1$, we have $\frac{(X_t+\Delta_t)}{e\mu} > \frac{1}{\lambda}$ and thus by Lemma 23 we have $\Pr[X_{t+1} \geq X_t + \Delta_t] \geq \frac{1}{4}$. At the same time by Chernoff bounds we have $\Pr[X_{t+1} < X_t] \leq \exp(-\frac{X'}{6})$. Hence, the drift of the potential function is at least $\frac{\min\{X', \Delta_{\min}\}}{4} - X' \exp(-\frac{X'}{6}) \geq \frac{X'}{8} - 3 \geq \frac{X'}{12}$.

Finally, applying the additive drift theorem (Theorem 2) we have

$$E[T(X')] \leq \frac{\Phi(X_0)}{\min\{\frac{1}{2}\Delta_{\min}, \frac{1}{12}X'\}} \leq \max\left\{\frac{4X' - 2X_0}{\Delta_{\min}}, 24\right\}. \qquad \square$$

Chapter 2 The Methods of Runtime Analysis of the Mutation-based Evolutionary Algorithms with Non-trivial Populations

As it was shown in Section 1.5, despite the impressive results of the last two decades, our understanding of the role of populations even in simple mutation-based EAs is still on unsatisfying level. The main problem which researchers face here is the complex dynamics of the populations. For the elitist $(\mu + \lambda)$ EA the only precise results we are aware of are for the special cases—the $(1 + \lambda)$ EA (where it is enough to track the dynamics of one parent individual) and the $(\mu + 1)$ EA (where the population dynamics are slow, namely only one individual can be changed in each iteration). For the non-elitist (μ, λ) EA there is more understanding of how non-trivial parent and offspring populations interact with each other, however, this has only being studied in two parameter settings when either algorithm fails to converge or when it replicates the behavior of the elitist $(\mu + \lambda)$ EA with high probability.

In this chapter we propose two analysis methods, the complete trees technique and methods of analysis of the no-drift processes. Then we show the effectiveness of these tools by finding the asymptotically precise runtime of the $(\mu + \lambda)$ EA and by studying the runtime of the (μ, λ) EA with the parameters which are on the threshold between the two previously observed regimes of the algorithm.

2.1 Proposed Analysis Methods

2.1.1 Complete Trees

The main problem when proving lower bounds for population-based algorithms is that many individuals which are created during the run of the EA are removed at some stage by selection operations. This creates a complicated population dynamics, which is very hard to follow via mathematical means.

One way to overcome this difficulty is to try to disregard the effect of selection and instead regard an optimistic version of the evolutionary process in which no individuals are removed. This idea can be traced back to [117]. In the context of evolutionary computation, it has been first used in [137] (see [136] for an extended version) in the analysis of a steady-state genetic algorithm with fitness-proportionate selection. In [85], this argument was used in the analysis of a $(\mu + 1)$ evolution strategy (in continuous search spaces). Not surprisingly, the analysis of the $(\mu + 1)$ EA [138] uses the artificial populations argument as well.

This technique then found applications in the analysis of memetic algorithms [129], aging-mechanisms [91], and non-elitist algorithms [96, 99]. The artificial population argument was also used to overcome the difficulties imposed by another removal mechanism, namely Pareto domination in evolutionary multi-objective optimization [46]. While similar in spirit, this work however uses quite different techniques, e.g., it does not represent the search process via tree structures.

Of course, to make the new process really an optimistic version of the original one, we have to ensure that, despite the larger population present, each individual which is also present in the true population has the same power of creating good solutions as in the original process. To ensure this in our process, we assume that in the artificial process each individual creates $\mathrm{Bin}(\lambda, 1/\mu)$ offspring. This assumption, in fact, leads to a much more drastic growth of the artificial population than the fact that we disregard selection.

When working with such an artificially enlarged population, there is a risk that the larger population finds it easier to create the optimal solution. This would give weaker lower bounds. So the main art in this proof approach is setting up the arguments in a way that the larger population does still, in an asymptotic sense, not find the optimum earlier than the original process. The reason why this is possible at all is that once selection is disregarded, the process consists only of independent applications of the mutation operator. This allows to use strong-concentration arguments which in the end give the desired result that none of the many members of the artificial population is the optimal solution.

To make this approach formal, we use the following notion of a *complete tree*, which, in simple words, describes all possible (iterated) offspring which could occur in a run of the evolutionary algorithm. This notion is different from those used in the works above, which all work with certain subtrees of the complete tree and use suitable arguments to reason that the restricted tree still covers all individuals that can, with reasonable probability, appear. We feel that our approach of working in the complete tree is technically simpler. For example, compared to [138], we do not first need to argue that with high probability the true tree has only certain depths and then, conditional on this event, argue that it does not contain an optimal solution. Working in the complete tree, we also do not need arguments from branching processes as used in [99]. Of course, the key argument that without selection we only do repeated unguided mutation, is used by us in the same flavor as in all previous works.

More precisely, the complete tree with initial individual x_0 is defined recursively as follow. Every vertex is labeled with some individual (a bit-string) which could potentially occur in the evolution process. The labels are not necessarily unique, but every vertex, except the root vertex v_0 is uniquely defined by the tuple (v,t,i), where v is the parent vertex (that is either the root vertex, or another vertex defined by a tuple), $t \in \mathbb{N}$ is the iteration when this vertex was created and $i \in [1..\lambda]$ is the number of the vertex among the vertices with the same v and t. The tree $T_0 = (V_0, E_0)$ at time $t = 0$ consists of the single (root) vertex v_0 that is labeled with the bit-string $c(v_0) = x_0$. Hence $E_0 = \emptyset$. If $T_t = (V_t, E_t)$ is defined for some $t \geq 0$, then we define the tree $T_{t+1} = (V_{t+1}, E_{t+1})$ as follows. For each vertex in V_t, we add λ vertices, connect them to this vertex, and generate their labels via standard-bit mutation from the parent. More precisely, let $N_{t+1} := \{(v_t, t+1, i) \mid v_t \in V_t, i \in [1..\lambda]\}$ and $V_{t+1} = V_t \cup N_{t+1}$. We call v_t the parent of $(v_t, t+1, i)$ and (v_t, t, i) the i-th child of v_t in iteration $t+1$. We generate the label $c(v_t, t+1, i)$ by applying standard-bit mutation to $c(v_t)$. We connect each new vertex with its parent, that is, we define $E_{t+1} = E_t \cup \{(v_t, (v_t, t+1, i)) \mid v_t \in V_t, i \in [1..\lambda]\}$. A simple example of a complete tree structure is shown in Figure 5.

It is easy to see that a complete tree at time t contains exactly $(\lambda+1)^t$ nodes, since each vertex from V_t has exactly λ new children in V_{t+1}. As said earlier, it thus massively overestimates the size of the true population of the EA.

For our purposes, it is not so much the total size of the tree that is important, but rather the number of nodes in a certain distance from the root. We estimate these in the following elementary lemma. Here and in the remainder, by *distance* we mean the graph theoretic distance,

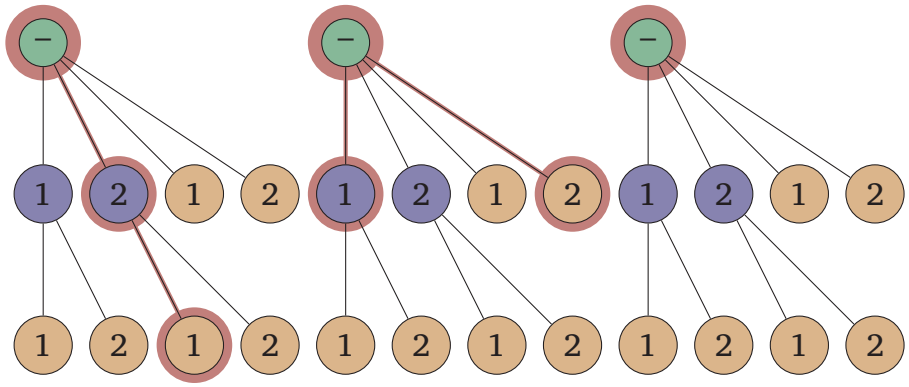


Figure 5 – *The structure of the complete tree for the* $(3+2)$ *EA after* $t = 2$ *iterations. The green vertices are the initial vertices, the blue vertices were created in the first iteration and the orange vertices were created in the second iteration. Each vertex is uniquely defined by a tuple of its parent vertex v, iteration it was created t and its number i among the children of its parent vertex created at the same iteration (the vertices in the figure are labeled with this number i). The highlighted vertices are the ones which were actually created by the algorithm. The labels are omitted in this illustration for reasons of readability*

that is, the length of the (in this case unique) path between the two vertices. Observe that this can be different from the iteration in which a node was generated. For example, the vertex (v_0, t, i), which is generated in iteration t from the initial vertex, has distance one from v_0.

Lemma 30. *Let* T_t *be a complete tree at time t. Let* $\ell \in \mathbb{N}_0$. *Then* T_t *contains exactly*

$$\binom{t}{\ell}\lambda^\ell$$

nodes in distance exactly ℓ *from the root.*

Proof. If $t < \ell$, then there are no vertices in distance ℓ (recall that in our notation $\binom{t}{\ell} = 0$ in this case). Otherwise, let v be a vertex in distance exactly ℓ from the root. Then there are times $1 \leq t_1 < \cdots < t_\ell \leq t$ and offspring numbers $i_1, \dots, i_\ell \in [1..\lambda]$ such that with the recursive definition of the vertices $v_1, \dots, v_\ell$ via $v_d = (v_{d-1}, t_d, i_d)$ for all $d \in [1..\ell]$, we have $v = v_\ell$. Hence, there are at most $\binom{t}{\ell}\lambda^\ell$ vertices in distance ℓ from the root. Conversely, each tuple of times and offspring numbers as above defines a different vertex in distance ℓ. Hence, there are at least $\binom{t}{\ell}\lambda^\ell$ different vertices in distance ℓ from the root. □

Clearly, a run of the $(\mu+\lambda)$ EA creates a subforest of μ disjoint complete trees with random root labels (complete forest). Whether a node of the complete forest appears in the forest describing the run of the $(\mu+\lambda)$ EA (the forest of the family trees) depends on the node labels (more precisely, on their fitness). However, regardless of the node labels the following is true: If some node v_s is present in the population at iteration t, then the edge $(v_s, (v_s, t, i))$ is present in the subforest at most with probability $1/\mu$, because for this it is necessary that the i-th offspring generated in iteration t chooses v_s as parent. Consequently, regardless of the nodes labels, the probability that a node in distance ℓ from the root in the complete forest enters the population of the $(\mu+\lambda)$ EA, is at most $\mu^{-\ell}$. Since we have not taken into account the node labels, we observe that the probability that a particular node of the complete forest (i) is labeled

with the optimum and (ii) makes it into the population of the $(\mu+\lambda)$ EA, is at most $\mu^{-\ell}p(\ell,n)$ with $p(\ell,n)$ denoting the probability that a vertex in distance ℓ from root is labeled with an optimum (we show this probability for the $(\mu+\lambda)$ EA with the standard bit mutation later in Lemma 39).

By Lemma 30, using a union bound over all nodes in the complete forest up to iteration t, we obtain the following theorem, which subsumes the method of the complete trees.

Theorem 6. *The probability that the $(\mu+\lambda)$ EA finds a unique optimum of a pseudo-Boolean in less than t iterations is*

$$q_{opt} \leq \mu \sum_{\ell=0}^{t} \binom{t}{\ell} \left(\frac{\lambda}{\mu}\right)^{\ell} p(\ell,n). \tag{3}$$

2.1.2 Drift Analysis for Processes with No Drift

In this section we consider stochastic processes with a very small drift compared to their random fluctuations. Such processes are often analyzed via different mathematical tools for martingales, e.g., with Azuma-Hoeffding inequality (Theorem 1.10.30 in [30]). However, all such tools aim at giving an upper bound on the probability that a martingale deviates from its expectation by some particular value. In this work we aim at the opposite result: we want to know how much time such process needs to deviate from its expectation by some value.

We propose to use drift analysis to estimate this time t. To do so, we make a transformation of the process. This idea has already been used to prove different drift theorems (e.g., variable or multiplicative) via the additive drift theorem. However, there it was used for the processes which already have drift, and the transformation only scaled the state space so that the drift was approximately the same in all possible states. In contrast with these previous works, we do a transformations which lets us obtain a drift from the variance of the original process. To illustrate the main idea of this transformation, consider a stochastic process $\{X_t\}_{t\in\mathbb{N}}$ such that

$$X_{t+1} = \begin{cases} X_t + 1 & \text{with probability } \frac{1}{2}, \\ X_t - 1 & \text{with probability } \frac{1}{2}. \end{cases}$$

This process has no drift, since $E[X_{t+1} \mid X_t] = \frac{1}{2}(X_t - 1) + \frac{1}{2}(X_t + 1) = X_t$. However, consider a function

$$h(x) = x(n - \ln(x) + 1)$$

and a transformed process $Y_t = h(X_t)$ (temporarily ignore that it is not defined for all X_t). Function $h(x)$ is monotonically increasing and concave in interval $(0, e^n)$, therefore if X_t is in this interval, then by Jensen's inequality we have

$$E[h(X_{t+1}) \mid X_t] = \frac{1}{2}h(X_t - 1) + \frac{1}{2}h(X_t + 1) < h(X_t).$$

Hence, the transformed process has a drift towards zero. This idea is illustrated in Figure 6

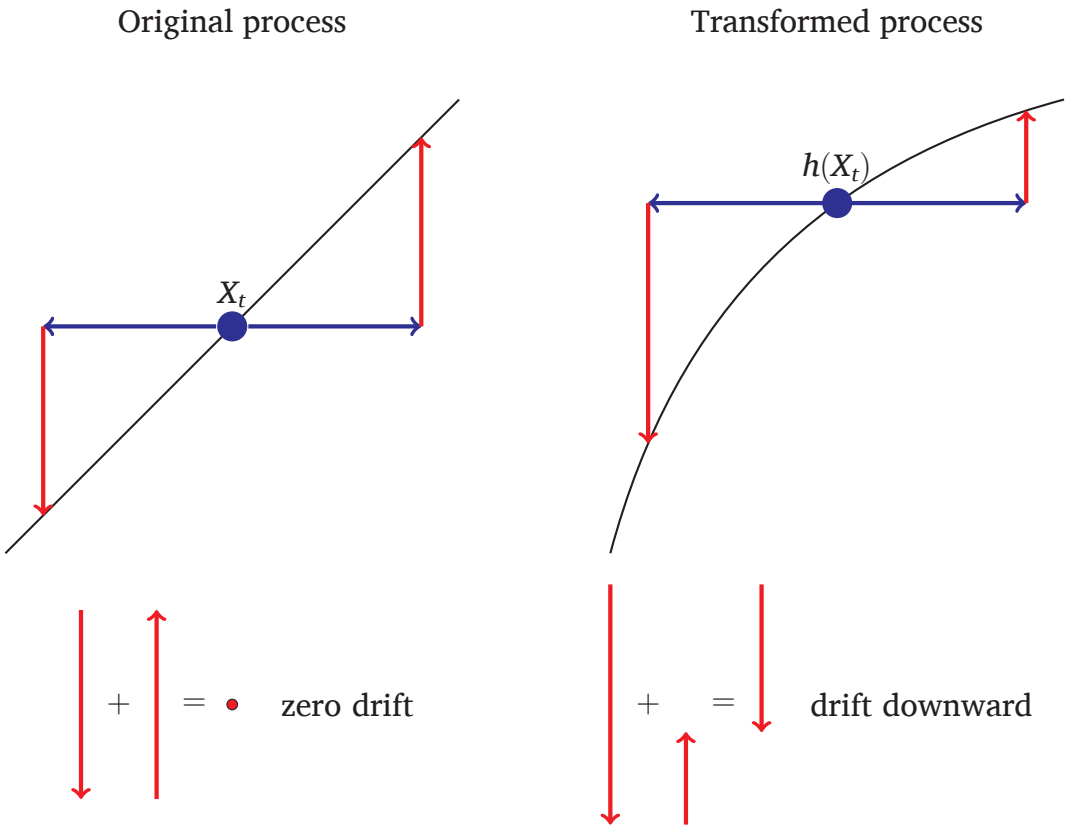


Figure 6 – *Illustration of the idea of how to transform a process with no drift to a process with drift.*

In particular, we apply such transformation to a sequence of random variables $\{X_t\}_{t\in\mathbb{N}}$ which represents the number of the best individuals in the population of the (μ,λ) EA. Unless the algorithm generates a better individual, the random variable X_{t+1} has a distribution very close to $\max\{\mathrm{Bin}(\lambda, \frac{X_t}{e\mu}), \mu\}$. If we have $\lambda \approx e\mu$, then $E[X_{t+1} - X_t]$ is very close to zero. Our aim is to find the first time t when X_t turns into zero. We show this time in the following theorem.

Theorem 7. *Let n be some positive integer number which tends to positive infinity (the Θ and big O notation is used relative to n). Let $\lambda, \mu \in \mathbb{N}$ satisfy*

1) $\lambda \leq e\mu$,
2) $\lambda \geq (1 - O(\frac{1}{\sqrt{n}}))e\mu$,
3) λ and μ are both $\omega(1)$ (which is necessary to satisfy the previous two conditions),
4) there exists $c = \Theta(1)$ such that $\mu \leq n^{\frac{1}{2}-c}$.

Let $\{X_t\}_{t\in\mathbb{N}}$ be a sequence of non-negative integer random variables, such that for all $t \in \mathbb{N}$ we have $X_t \sim \min\{\mu, \mathrm{Bin}(\lambda, p_n(t))\}$ with $p_n(t)$ close to $\frac{X_{t-1}}{e\mu}$, namely

$$\left| p_n(t) - \frac{X_t}{e\mu} \right| \leq \frac{2X_t}{e\mu\sqrt{n}}.$$

Let also X_t be zero with probability one, if $X_{t-1} = 0$. Then there exist $s = \Theta(1)$ and $\alpha = (80\mu + 1)$ such that $X_\alpha = 0$ with probability at least s.

Before we prove this theorem we show the following lemma, where we transform the process X_t and show that the transformed process has a constant drift towards zero.

Lemma 31. *Let X_t be a process from Theorem 7. Let $h(x) = x(\ln(\mu) - \ln(x) + 2)$ for all $x \in [1..\lambda]$ and let $h(0) = 0$. Let $\beta = \frac{1}{20}$. Then there exits a constant S such that*

$$E[h(X_t) - h(X_{t+1}) \mid X_t \geq S] \geq \beta.$$

Proof. Assume that $X_t = x$ for some $x \in [1..\mu]$. Since we only consider one moment in time t, let us denote $p_n(t+1)$ by p_n for brevity. Let $\tilde{X}_{t+1}$ be a random variable which follows distribution $\mathrm{Bin}(\lambda, p_n)$ and which therefore dominates X_{t+1}. Since h is monotonically increasing, we have

$$\begin{aligned} E[h(X_t) - h(X_{t+1}) \mid X_t = x] &\geq E[h(X_t) - h(\tilde{X}_{t+1}) \mid X_t = x] \\ &= E\left[x(\ln\mu - \ln x + 2) - \tilde{X}_{t+1}(\ln\mu - \ln\tilde{X}_{t+1} + 2) \mid X_t = x\right] \\ &= (\ln\mu + 2)E[x - \tilde{X}_{t+1} \mid X_t = x] + E[\tilde{X}_{t+1}\ln\tilde{X}_{t+1} \mid X_t = x] - x\ln x. \end{aligned} \tag{4}$$

We first show that the first term has a very small value. Since $\tilde{X}_{t+1} \sim \mathrm{Bin}(\lambda, p_n)$, its expectation is λp_n. Hence, we have

$$\begin{aligned} (\ln\mu + 2)E[x - \tilde{X}_{t+1} \mid X_t = x] &= (\ln\mu + 2)(x - \lambda p_n) \\ &\geq (\ln\mu + 2)\left(x - \frac{\lambda x}{e\mu}\left(1 + \frac{2}{\sqrt{n}}\right)\right) \\ &\geq (\ln\mu + 2)\left(x - x\left(1 + \frac{2}{\sqrt{n}}\right)\right) \\ &\geq -(\ln\mu + 2)\frac{x}{\sqrt{n}} \\ &\geq -(\ln\mu + 2)\frac{\mu}{\sqrt{n}} \\ &\geq -\left(\left(\frac{1}{2} - c\right)\ln(n) + 2\right)n^{\frac{1}{2}-c-\frac{1}{2}} \\ &\geq -O\left(n^{-\frac{c}{2}}\right) = -o(1). \end{aligned} \tag{5}$$

Since $\tilde{X}_{t+1}$ follows a binomial distribution, it can be represented as a sum of λ Bernoulli random variables $\{Y_i\}_{i\in[1..\lambda]}$ with success probability p_n. Thus, we have

$$\begin{aligned} E[\tilde{X}_{t+1}\ln\tilde{X}_{t+1} \mid X_t = x] &= E\left[\sum_{i=1}^{\lambda} Y_i \ln\left(\sum_{j=1}^{\lambda} Y_j\right) \mid X_t = x\right] \\ &= \sum_{i=1}^{\lambda} E\left[Y_i \ln\left(\sum_{j=1}^{\lambda} Y_j\right) \mid X_t = x\right] \\ &= \lambda p_n E\left[\ln\left(1 + \sum_{j=1}^{\lambda-1} Y_j\right) \mid X_t = x\right]. \end{aligned}$$

Let $X' \coloneqq \sum_{j=1}^{\lambda-1} Y_j$, then it follows a binomial distribution $\mathrm{Bin}(\lambda - 1, p_n)$. By Lemma 28, if λp_n is not less than some constant $S_{\min}$ then we have

$$E[\ln(1 + X')] = \ln(1 + (\lambda - 1)p_n) - \frac{5}{6}\frac{(1 - p_n)}{(\lambda - 1)p_n}.$$

Hence, we compute

$$\begin{aligned} E[\tilde{X}_{t+1} \ln \tilde{X}_{t+1} \mid X_t = x] - x \ln x & \\ &\geq \lambda p_n \left(\ln(1+\lambda p_n) - \frac{5}{6} \frac{(1-p_n)}{\lambda p_n} \right) - x \ln x \\ &= \lambda p_n \left(\ln(1+\lambda p_n) - \frac{5}{6} \frac{(1-p_n)}{\lambda p_n} \right) - \lambda p_n \ln(\lambda p_n) \\ &+ \lambda p_n \ln(\lambda p_n) - x \ln x \end{aligned} \tag{6}$$

First we note that since $x \mapsto x \ln x$ is a continuous function and since $\lambda p_n = \frac{\lambda x}{e\mu}(1 \pm O(\frac{1}{\sqrt{n}})) = x(1 \pm O(\frac{1}{\sqrt{n}}))$, we have

$$\begin{aligned} \lambda p_n \ln(\lambda p_n) - x \ln x &\geq -xO\left(\frac{1}{\sqrt{n}}\right) \geq -\mu O\left(\frac{1}{\sqrt{n}}\right) \\ &\geq -n^{\frac{1}{2}-c} O\left(\frac{1}{\sqrt{n}}\right) \geq -o(1). \end{aligned}$$

Recall that $\lambda = \omega(1)$ and that p_n satisfies the following bounds

$$\begin{aligned} p_n &\geq \frac{x}{e\mu}\left(1 - \frac{2}{\sqrt{n}}\right), \\ p_n &\leq \frac{x}{e\mu}\left(1 + \frac{2}{\sqrt{n}}\right) \leq \frac{1}{e} + o(1). \end{aligned}$$

Thus, we have

$$\begin{aligned} \lambda p_n \left(\ln(1+\lambda p_n) - \frac{5}{6} \frac{(1-p_n)}{\lambda p_n} \right) - \lambda p_n \ln(\lambda p_n) & \\ &= \lambda p_n \left(\ln\left(1 + \frac{1-p_n}{\lambda p_n}\right) - \frac{5}{6} \frac{(1-p_n)}{\lambda p_n} \right) \\ \text{(by Lemma 18)} &\geq \lambda p_n \left(\frac{1-p_n}{\lambda p_n} - 2\left(\frac{1-p_n}{\lambda p_n}\right)^2 - \frac{5}{6} \cdot \frac{1-p_n}{\lambda p_n} \cdot \frac{\lambda}{\lambda - 1} \right) \\ &\geq (1-p_n)\left(1 - \frac{5\lambda}{6(\lambda-1)} - 2\frac{(1-p_n)}{\lambda p_n}\right) \\ \text{(since } \lambda = \omega(1)) &\geq \left(1 - \frac{1}{e} - o(1)\right)\left(1 - \frac{5}{6} - \frac{2}{x} - o(1)\right) \\ \text{(if } x \geq 24) &\geq \left(1 - \frac{1}{e}\right)\left(\frac{1}{6} - \frac{1}{12}\right) - o(1) \geq \frac{e-1}{12e} - o(1). \end{aligned}$$

Putting this into (6) and recalling (5) and (4), we conclude that if $X_t \geq S := \max\{S_{\min}, 24\}$, then we have

$$E[h(X_t) - h(X_{t+1}) \mid X_t \geq S] \geq \frac{e-1}{12e} - o(1) \geq \frac{1}{20} =: \beta$$

□

Finally, we prove Theorem 7.

Proof of Theorem 7. Let S be a constant from Lemma 31. We define potential function

$$h(x) = \begin{cases} x(\ln\mu - \ln x + 2), & \text{if } x \geq S \\ 0, & \text{otherwise.} \end{cases}$$

Since we only lowered the potential values for $x \leq S$, Lemma 31 holds also for this potential function. Therefore, by Lemma 31 and by the additive drift theorem (Theorem 2) the expected number of generations $E[\tau]$ until X_t becomes less than S is at most

$$E[\tau] \leq 20E[h(X_0)] \leq 20h(\mu) = 40\mu.$$

By Markov's inequality the probability that X_t does not reach S in less than 80μ generations is at most $\frac{1}{2}$.

As soon as X_t becomes less than S, the probability that in the next generation X_t becomes zero is

$$(1-p_n)^\lambda \geq \left(1 - \frac{S}{e\mu}(1-o(1))\right)^\lambda = \left(1 - \frac{S}{\lambda}(1-o(1))\right)^\lambda \xrightarrow{\lambda\to+\infty} e^{-S}.$$

Hence, if λ is large enough, this probability is at least $\frac{2e^{-S}}{3}$.

Finally, the probability that process X_t reaches zero in less than $\alpha = 80\mu + 1$ generation is at least $s := \frac{e^{-S}}{3} = \Theta(1)$. Since as long as X_t reaches zero it cannot increase, we have $X_\alpha = 0$ with the same probability. □

2.2 Analysis of the $(\mu+\lambda)$ EA

In this section we prove that for arbitrary values of μ and λ (which can be functions of the problem size n, however, for the lower bound we assume that μ is at most polynomial in n), the expected number of iterations the $(\mu+\lambda)$ EA takes to find the optimum of the OneMax function is

$$E[T] = \Theta\left(\frac{n\log n}{\lambda} + \frac{n}{\lambda/\mu} + \frac{n\log^+\log^+(\lambda/\mu)}{\log^+(\lambda/\mu)}\right),$$

where $\log^+ x := \max\{1, \log x\}$ for all $x > 0$. This result subsumes the previous results for the $(1+\lambda)$ EA and $(\mu+1)$ EA obtained in [87, 49, 138].

This runtime guarantee shows, e.g., that using a true parent population of size at most $\max\{\log n, \lambda\}$ does not reduce the asymptotic runtime compared to $\mu = 1$. Such information can be useful since it is known that larger parent population sizes can increase the robustness to noise, see, e.g., [74].

With our methods, we can also analyze a related algorithm. He and Yao [80] and Chen, He, Sun, Chen, and Yao [13] analyzed the $(\lambda \overset{1:1}{+} \lambda)$ EA. We prove a tight runtime bound of $\Theta(\frac{n\log n}{\lambda} + n)$ iterations, which also shows that the fairness in the parent selection does not change the asymptotic runtime in this problem.

2.2.1 Upper Bounds

In this section, we prove separately two upper bounds for the runtime of the $(\mu+\lambda)$ EA on the ONEMAX problem, the first one being valid for all values of μ and λ and the second one giving an improvement for the case that λ is large compared to μ, more precisely, that $\lambda/\mu \geq e^e$.

Although we do not propose novel analysis methods in this subsection, it is important to have these upper bounds to show the tightness of our lower bound which we obtain via the complete trees technique and to better understand the dynamics of the population of the $(\mu+\lambda)$ EA.

Where not specified differently, we denote the current best fitness in the population by i and the number of best individuals in the population by j.

Increase of the Number of the Best Individuals

In this subsection we analyze how the number of individuals on the current-best fitness level increases over time and derive from this two estimates for the time taken for a fitness improvement. We note that often it is much easier to generate an additional individual with current-best fitness by copying an existing one than to generate an individual having strictly better fitness by flipping the right bits. Consequently, in a typical run of the $(\mu+\lambda)$ EA, first the number of best individuals will increase to a certain number and only then it becomes likely that a strict improvement happens.

Since the increase of the number of individuals on the current-best fitness level via producing copies of such best individuals is independent of the fitness function, we formulate our results for the optimization of an arbitrary pseudo-Boolean function and hope that they might find applications in other runtime analyses as well. So let $f : \{0,1\}^n \to \mathbb{R}$ be an arbitrary fitness function which we optimize using the $(\mu+\lambda)$ EA.

Assume that the $(\mu+\lambda)$ EA starts in an arbitrary state where the best individuals have fitness i and there are j_1 such individuals in the population. At this point due to the elitism the algorithm cannot decrease the best fitness i and it also cannot decrease the number of the best individuals j_1 until it increases the best fitness. Following [129, Lemma 2] we call an individual *fit* if it has a fitness i or better. For $j_2 \in \mathbb{N}$, we define $\tau_{j_1,j_2}(i)$ to be the first time (number of iterations) at which the population of the $(\mu+\lambda)$ EA contains at least j_2 fit individuals. We note that this random variable $\tau_{j_1,j_2}(i)$ may depend on the particular initial state of the $(\mu+\lambda)$ EA, but since our results are independent of this initial state (apart from i and j_1) we suppress in our notation the initial state.

The time $\tau_{1,\mu}(i)$, that is, the specific case that $j_1 = 1$ and $j_2 = \mu$, is also called the *takeover time* of a new best individual. For this takeover time, Sudholt [129, Lemma 2] proved the upper bound

$$E[\tau_{1,\mu}(i)] = \lceil \log_5 \mu \rceil \left(\frac{32}{1-\frac{1}{e}} \cdot \frac{\mu}{\lambda} + 1 \right) = O\left(\frac{\mu \log \mu}{\lambda} + \log \mu \right) \tag{7}$$

for any $i \in [0..n-1]$.

In this section we improve this result by (i) treating the general case of arbitrary $j_1, j_2 \in [1..\mu]$ and (ii) by showing an asymptotically smaller bound for the case $\lambda = \omega(\mu)$. In our main analysis of the $(\mu+\lambda)$ EA, we need takeover times for general values of j_2 to profit from the event when we get a fitness gain before the population contains only best individuals, which is likely to happen on the lower fitness levels as we show further. The extension to general values of j_1 is not needed, but since it does not take extra effort, we do it on the way.

We first prove the following result for arbitrary values of μ and λ. We need this result since it allows arbitrary target numbers j_2.

Lemma 32. *Let $i \in [0..n-1]$ and $j_1, j_2 \in [1..\mu]$ with $j_1 < j_2$. Then*

$$E[\tau_{j_1 j_2}(i)] \leq \frac{2e\mu}{\lambda}\left(\ln\frac{j_2}{j_1}+1\right) + (j_2 - j_1).$$

Proof. To prove this lemma we use Theorem 3. For this purpose we define levels $A_{j_1}, \ldots, A_{j_2}$. For any $j \in [j_1..j_2-1]$ the populations in level A_j have exactly j fit individuals. The level A_{j_2} consists of all populations with at least j_2 fit individuals. Note that the $(\mu+\lambda)$ EA cannot go from level A_j to any other level with smaller index, since it cannot decrease the number of fit individuals due to the elitist selection.

If there are j fit individuals in the population, then the probability $p_1(j)$ to create as one offspring a copy of a fit individual is the probability to select one of j fit individuals as a parent multiplied by the probability not to flip any bit of it during the mutation. Hence,

$$p_1(j) \geq \frac{j}{\mu}\left(1-\frac{1}{n}\right)^n \geq \frac{j}{2e\mu}, \tag{8}$$

where we used the inequality $(1-\frac{1}{n})^n \geq \frac{1}{2e}$ that holds for all $n \geq 2$.

The probability $p_2(j)$ to leave level A_j in one iteration is at least the probability to create a copy of a fit individual as one of the λ offspring. Hence, by Lemma 21 we have

$$p_2(j) \geq 1-(1-p_1(j))^\lambda \geq \frac{1}{1+\frac{1}{p_1(j)\lambda}} \geq \frac{1}{1+\frac{2e\mu}{j\lambda}}. \tag{9}$$

By Theorem 3 we have

$$E[\tau_{j_1 j_2}(i)] \leq \sum_{j=j_1}^{j_2-1}\frac{1}{p_2(j)} \leq \sum_{j=j_1}^{j_2-1}\left(1+\frac{2e\mu}{j\lambda}\right) \leq \frac{2e\mu}{\lambda}\left(\ln\frac{j_2}{j_1}+1\right) + (j_2-j_1).$$

□

We note that in case when $j_1 = 1$ and $j_2 = \mu$ our upper bound is $O(\frac{\mu\log\mu}{\lambda}+\mu)$. This is weaker than the upper bound (7) given in [129, Lemma 2] if $\lambda = \omega(\log\mu)$. Without proof we note that in all other cases the two bounds are asymptotically equal.

The reason that our bound is weaker in some cases is that we do not consider the event that the algorithm generates more than one fit offspring in one iteration, while Sudholt in [129, Lemma 2] proved that the number of the fit offspring is multiplied by some constant factor in every $32\mu/\lambda$ iterations. The same idea may be used to prove the bound

$$E[\tau_{j_1 j_2}(i)] \leq \left\lceil \log_5\frac{j_2}{j_1}\right\rceil\left(\frac{32}{1-\frac{1}{e}}\cdot\frac{\mu}{\lambda}+1\right) = O\left(\frac{\mu\log\frac{j_2}{j_1}}{\lambda}+\log\frac{j_2}{j_1}\right). \tag{10}$$

We still prefer to use to Lemma 32 in our proofs, since it gives us a bound that is easier to operate with due to the simpler leading constants of each term, while the greater terms do not affect our main results.

We now give a second bound for the case that $\frac{\lambda}{\mu} \geq e^e$. It is asymptotically stronger than (7) when $\lambda = \omega(\mu)$ and $\mu = \omega(1)$.

Lemma 33. *Let $\frac{\lambda}{\mu} \geq e^e$. Let $i \in [1..n-1]$ and $j_1, j_2 \in [1..\mu]$ with $j_1 < j_2$. Then*

$$E[\tau_{j_1,j_2}(i)] \leq 4\,\frac{\ln \frac{j_2}{j_1}}{\ln \frac{\lambda}{2e\mu}} + 4.$$

Proof. Let the current population have j fit individuals. Then by (8) the probability that a fixed offspring is a copy of a fit individual is $p_1(j) \geq \frac{j}{2e\mu}$. Therefore, the number N of fit individuals among the λ offspring dominates stochastically a random variable B with binomial distribution $\mathrm{Bin}\left(\lambda, \frac{j}{2e\mu}\right)$. We have $E[B] = \frac{\lambda j}{2e\mu}$. By Lemma 23, $\Pr[B \geq E[B]] \geq \frac{1}{4}$ and thus $\Pr[N \geq \frac{j}{2e\mu}] \geq \frac{1}{4}$. Consequently, in each iteration with probability at least $\frac{1}{4}$ the number of the fit individuals in the population is multiplied by a factor of at least $(1 + \frac{\lambda}{2e\mu})$ (but obviously it cannot become greater than μ).

For a formal proof we define the levels $A_1, \ldots, A_m$, where

$$m := \left\lceil \frac{\ln \frac{j_2}{j_1}}{\ln\left(1 + \frac{\lambda}{2e\mu}\right)} \right\rceil + 1.$$

Level A_m consists of the populations with at least j_2 fit individuals. For $k \in [1..m-1]$ the populations of level A_k have exactly j fit individuals, where

$$j \in \left[j_1 \left(1 + \frac{\lambda}{2e\mu}\right)^{k-1},\ j_1 \left(1 + \frac{\lambda}{2e\mu}\right)^{k} - 1 \right],$$

and $j < j_2$. To leave any level it is enough to multiply the number of the best individuals by $1 + \frac{\lambda}{2e\mu}$, and the probability of this event is at least $\frac{1}{4}$. By Theorem 3 we have

$$\begin{aligned} E[\tau_{j_1,j_2}(i)] &\leq \sum_{k=1}^{m-1} 4 = 4\left\lceil \frac{\ln \frac{j_2}{j_1}}{\ln\left(1 + \frac{\lambda}{2e\mu}\right)} \right\rceil \\ &\leq 4\,\frac{\ln \frac{j_2}{j_1}}{\ln \frac{\lambda}{2e\mu}} + 4. \end{aligned}$$

□

We note that the proof of Lemma 33 holds for the weaker assumption $\frac{\lambda}{\mu} > 2e$ as well. However in order not to confuse the reader later when we consider the case $\frac{\lambda}{\mu} > e^e$ and where this lemma is used, we formulate Lemma 33 with unnecessarily stronger condition.

When $j_2 = \mu$ and $j_1 = 1$ the bound yielded by Lemma 33 is at least as tight as that of (7). For the general values of j_1 and j_2 our bound is at least as tight as the bound (10). When $\lambda/\mu \geq e^e$ the bound (10) simplifies to $O(\log \frac{j_2}{j_1})$. If $\lambda = \omega(\mu)$ and $\frac{j_2}{j_1} = \omega(1)$ then we have

$$4\frac{\ln \frac{j_2}{j_1}}{\ln \frac{\lambda}{2e\mu}} + 4 = o\left(\log \frac{j_2}{j_1}\right).$$

Therefore, in this case the bound given in Lemma 33 is asymptotically smaller than (10). In all other cases the two bounds are asymptotically equal.

The reason that we have obtained a tighter bound is that we have proven that the number of the fit individuals is multiplied by a super-constant factor with constant probability, while the proof of [129, Lemma 2] considers only the multiplication by a constant factor.

We now use Lemmas 32 and 33 to prove estimates for the time it takes to obtain a strictly better individual once the population contains at least one individual of fitness i. We define $\tilde{T}_i$ as the number of iterations before the algorithm finds an individual with fitness greater than i, if it already has an individual with fitness i in the population. As before, this random variable depends on the precise initial state, but since our results do not rely on the initial state, we suppress it in this notation.

To prove upper bounds on $\tilde{T}_i$, we estimate the time it takes until some number $\mu_0(i) \in [1..\mu]$ of individuals with fitness at least i are in the population and then estimate the time to find an improving solution from this situation. We phrase our results here in terms of $\mu_0(i)$ and optimize the value of $\mu_0(i)$ in the later subsections.

Corollary 1. *For any $i \in [0..n-1]$ and $\mu_0(i) \in [1..\mu]$, we have*

$$E[\tilde{T}_i] \leq \mu_0(i) + \frac{2e\mu}{\lambda}(\ln(\mu_0(i)) + 1) + \frac{e\mu n}{\lambda(n-i)\mu_0(i)}.$$

Proof. Even if the algorithm has only one best individual in the population, in $\tau_{1,\mu_0(i)}(i)$ iterations it will have at least $\mu_0(i)$ individuals with fitness at least i. Assume that at this time we have no individuals with fitness better than i (since otherwise we are done). Let $\tau^+(i)$ be the runtime until the algorithm creates an individual with fitness at least $i+1$ if it already has at least $\mu_0(i)$ individuals with fitness i in the population.

In this setting the probability $p'(i)$ that a particular offspring has fitness better than i is at least the probability to choose one of the $\mu_0(i)$ best individuals and to flip only one of $n-i$ zero-bits in it. We estimate

$$p'(i) \geq \frac{\mu_0(i)(n-i)}{\mu n}\left(1-\frac{1}{n}\right)^{n-1} \geq \frac{(n-i)\mu_0(i)}{e\mu n}.$$

By Lemma 21 the probability $p''(i)$ to create at least one superior individual among the λ offspring is

$$p''(i) \geq 1-(1-p'(i))^\lambda \geq \frac{\lambda p'(i)}{1+\lambda p'(i)} = \frac{1}{1+\frac{1}{\lambda p'(i)}} \geq \frac{1}{1+\frac{e\mu n}{\lambda(n-i)\mu_0(i)}}. \tag{11}$$

With $p''(i)$ we estimate $E[\tau^+(i)] \leq \frac{1}{p''(i)}$. Therefore, by Lemma 32 we have

$$E[\tilde{T}_i] \leq E[\tau_{1,\mu_0(i)}(i) + \tau^+(i)] = E[\tau_{1,\mu_0(i)}(i)] + E[\tau^+(i)]$$

$$\begin{aligned}
&\leq E[\tau_{1,\mu_0(i)}(i)] + \frac{1}{p''(i)} \\
&\leq \frac{2e\mu}{\lambda}(\ln \mu_0(i) + 1) + (\mu_0(i) - 1) + 1 + \frac{e\mu n}{\lambda(n-i)\mu_0(i)} \\
&= \mu_0(i) + \frac{2e\mu}{\lambda}(\ln(\mu_0(i)) + 1) + \frac{e\mu n}{\lambda(n-i)\mu_0(i)}.
\end{aligned}$$

□

Corollary 2. *If $\frac{\lambda}{\mu} > e^e$ then for any $i \in [0..n-1]$ and $\mu_0(i) \in [1..\mu]$, we have*

$$E[\tilde{T}_i] \leq 4\frac{\ln \mu_0(i)}{\ln \frac{\lambda}{2e\mu}} + \frac{e\mu n}{\lambda(n-i)\mu_0(i)} + 5.$$

Proof. Using the same arguments as in the proof of Corollary 1 (in particular, the estimate for $p''(i)$ given in (11)) and by Lemma 33 we estimate

$$\begin{aligned}
E[\tilde{T}_i] &\leq E[\tau_{1,\mu_0(i)}(i) + \tau^+(i)] = E[\tau_{1,\mu_0(i)}(i)] + E[\tau^+(i)] \\
&\leq E[\tau_{1,\mu_0(i)}(i)] + \frac{1}{p''(i)} \\
&\leq 4\frac{\ln \mu_0(i)}{\ln \frac{\lambda}{2e\mu}} + 4 + 1 + \frac{e\mu n}{\lambda(n-i)\mu_0(i)} \\
&= 4\frac{\ln \mu_0(i)}{\ln \frac{\lambda}{2e\mu}} + \frac{e\mu n}{\lambda(n-i)\mu_0(i)} + 5.
\end{aligned}$$

□

We note that Lemma 33 is tight in the sense that we cannot obtain a better upper bound using only the argument of copying the fit individuals.

To formalize this we assume that there is a set $D \subseteq \{0,1\}^n$ of *desired individuals*. We regard the variant EA^0 of the $(\mu+\lambda)$ EA which only accepts offspring which are desired individuals identical to their parent. Note that the number of desired individuals in a run of this artificial algorithm can never decrease. Assuming that the initial population of the EA^0 contains exactly j_1 desired individuals, we define $\tau^*_{j_1 j_2}(i)$ as the number of iterations until the population of the EA^0 contains at least j_2 desired individuals (unlike before, this notation does not depend on the precise initial population as long as it has exactly j_1 desired individuals, but this fact is not important in the following). We show the following result.

Lemma 34. *Let $\frac{\lambda}{\mu} \geq e^e$. Let j_1, j_2 be some integer numbers in $[1..\mu]$ such that $j_2 > j_1$. Then*

$$E[\tau^*_{j_1 j_2}(i)] = \Omega\left(\frac{\log \frac{j_2}{j_1}}{\log \frac{\lambda}{\mu}} + 1\right).$$

Proof. If $\frac{j_2}{j_1} \leq \frac{\lambda}{\mu}$, then

$$\frac{\log \frac{j_2}{j_1}}{\log \frac{\lambda}{\mu}} + 1 = \Theta(1)$$

and the claim is trivial, since we need at least one iteration to increase the number of the copies in the population.

Consider $\frac{j_2}{j_1} \geq \frac{\lambda}{\mu} \geq e^e$. It is enough to show that $E[\tau^*_{j_1 j_2}(i)] = \Omega(\log(\frac{j_2}{j_1})/\log(\frac{\lambda}{\mu}))$ (without $+1$ term). Let $j(t)$ be the number of the desired individuals after iteration t. We have $j(0) = j_1$. Let $N(t)$ be the number of desired individuals newly created in iteration t. Then $N(t)$ follows a binomial law $\mathrm{Bin}(\lambda, \frac{j(t-1)}{e_n \mu})$, where $e_n := (1-\frac{1}{n})^{-n} \geq e$. Hence, we have $E[N(t) \mid j(t-1)] = \frac{j(t-1)\lambda}{e_n\mu} \leq \frac{j(t-1)\lambda}{e\mu}$.

For any $t \in \mathbb{N}$ we have $j(t) \leq j(t-1) + N(t)$, where strict inequality occurs only if $j(t-1) + N(t) > \mu$. Therefore, we have

$$\begin{aligned} E[j(t)] &= E[E[j(t) \mid j(t-1)]] \leq E[E[j(t-1) + N(t) \mid j(t-1)]] \\ &= E[j(t-1)] + E[E[N(t) \mid j(t-1)]] \leq E[j(t-1)] + E\left[\frac{j(t-1)\lambda}{e\mu}\right] \\ &= E[j(t-1)] + \frac{\lambda}{e\mu} E[j(t-1)] = \left(1 + \frac{\lambda}{e\mu}\right) E[j(t-1)]. \end{aligned}$$

By induction, we obtain

$$E[j(t)] \leq \left(1 + \frac{\lambda}{e\mu}\right)^t j(0) = \left(1 + \frac{\lambda}{e\mu}\right)^t j_1,$$

and by Markov's inequality, we have

$$\Pr[j(t) \geq j_2] \leq \frac{E[j(t)]}{j_2} \leq \left(1 + \frac{\lambda}{e\mu}\right)^t \frac{j_1}{j_2}.$$

For

$$t := \frac{\ln \frac{j_2}{2j_1}}{\ln\left(1 + \frac{\lambda}{e\mu}\right)} = \Omega\left(\frac{\log \frac{j_2}{j_1}}{\log \frac{\lambda}{\mu}}\right)$$

we obtain

$$\Pr[j(t) \geq j_2] \leq \frac{1}{2}.$$

Hence, the probability that the EA^0 does not obtain j_2 desired individuals in $t = \Omega(\frac{\log(j_2/j_1)}{\log(\lambda/\mu)})$ iterations is at least constant. Thus, the expected number of iterations before this happens is at least $\Omega(\frac{\log(j_2/j_1)}{\log(\lambda/\mu)})$. □

Unconditional Upper Bound

Having the results of the previous subsection for the takeover times we first prove the following upper bound, which is valid for all values of μ and λ. When λ is not significantly larger than μ, then the $(\mu+\lambda)$ EA typically increases the best fitness by at most a constant in each iteration. For this reason, we can use Theorem 3 and obtain a runtime bound that will turn out to be tight for this case.

Theorem 8. *The expected number of iterations for the $(\mu+\lambda)$ EA to optimize the* ONEMAX *problem is*

$$O\left(\frac{n\log n}{\lambda}+\frac{n\mu}{\lambda}+n\right).$$

Proof. To use Theorem 3 we define levels $A_0,\dots,A_n$ such that level A_i, $i\in[0..n]$, consists of all populations having maximum fitness equal to i. In Corollary 1 we have already estimated the expected times $E[\tilde{T}_i]$ the $(\mu+\lambda)$ EA takes to leave these levels. These estimates depended on the number $\mu_0(i)$ of individuals of fitness i we aim at before leaving the level. By choosing suitable values for the $\mu_0(i)$ we prove our bound.

The choice of $\mu_0(i)$ is guided by the following trade-off. If we choose $\mu_0(i)=\mu$, then after having μ best individuals in the population we have the highest probability to find a better individual. However, we pay for it with the time we spend on obtaining μ copies of the best individual. On the other hand, if we choose $\mu_0(i)=1$ we do not spend any iteration filling the population with copies of the best individual, but we have a low chance to increase the current fitness. How this trade-off is optimally resolved, and hence the optimal value of $\mu_0(i)$, depends on the probability to create a better individual and thus on the current fitness i.

We distinguish three cases depending on current fitness i. The "milestones" which mark the transition between these cases are the fitness values $i=\lceil n-\frac{n}{2+\lambda/(e\mu)}\rceil$ and $i=\lfloor n-\frac{n}{\mu(2+\lambda/e)}\rfloor$. While the best fitness is below the first milestone, the probability to increase the fitness is so high that we do not need to have more than one best individual in the population. Beyond the second milestone this probability is so low that we better spend the time to fill the population with the copies of the best individual. Between the two milestones we have to find a suitable value of $\mu_0(i)$ to give a balanced trade-off.

To simplify the notation, we define $\Delta_i := 1+\sqrt{1+\frac{n\lambda}{e(n-i)\mu}}$. Note that

$$1+\sqrt{\frac{n}{n-i}}\sqrt{\frac{\lambda}{e\mu}}\le\Delta_i\le 1+\sqrt{\frac{n}{n-i}}\sqrt{1+\frac{\lambda}{e\mu}}. \tag{12}$$

This value of Δ_i arises from the computation of the derivative of the upper bound on $E[\tilde{T}_i]$ from Corollary 1, which is needed to find the optimal value of $\mu_0(i)$ in the second case, when the current fitness is between the two milestones.

For $i\le\lceil n-\frac{n}{2+\lambda/(e\mu)}\rceil$ we define $\mu_0(i):=1$. By Corollary 1 we have

$$E[\tilde{T}_i]\le\frac{e\mu n}{\lambda(n-i)}+\frac{2e\mu}{\lambda}+1.$$

Let T_1 be the number of iterations before the $(\mu+\lambda)$ EA finds an individual with fitness greater than $\lceil n-\frac{n}{2+\lambda/(e\mu)}\rceil$ for the first time. Then by Theorem 3 we have

$$\begin{aligned}E[T_1]&\le\sum_{i=0}^{\lceil n-\frac{n}{2+\lambda/e\mu}\rceil}E[\tilde{T}_i]\le\sum_{i=0}^{\lceil n-\frac{n}{2+\lambda/e\mu}\rceil}\left(\frac{e\mu n}{\lambda(n-i)}+\frac{2e\mu}{\lambda}+1\right)\\&\le\frac{e\mu n}{\lambda}\left(\ln(n)-\ln\left(\frac{n}{2+\lambda/e\mu}\right)+1\right)+\frac{2e\mu}{\lambda}n+n\\&=\frac{e\mu n}{\lambda}\ln\left(2+\frac{\lambda}{e\mu}\right)+\frac{3e\mu n}{\lambda}+n\end{aligned}$$

$$= O\Big(\frac{\mu n}{\lambda}\Big) + O(n),$$

where we used the estimate $\frac{e\mu}{\lambda}\ln(2+\frac{\lambda}{e\mu}) = O(1+\frac{\mu}{\lambda})$ that holds for any asymptotic behavior of μ/λ.

For $\lceil n - \frac{n}{2+\lambda/(e\mu)}\rceil < i \le \lfloor n - \frac{n}{\mu(2+\lambda/e)}\rfloor$ we define $\mu_0(i) := \lceil \frac{n}{(n-i)\Delta_i}\rceil$. By Corollary 1 we have

$$E[\tilde{T}_i] \le \frac{n}{(n-i)\Delta_i} + 1 + \frac{2e\mu}{\lambda}\left(\ln\frac{n}{(n-i)\Delta_i} + 2\right) + \frac{e\mu\Delta_i}{\lambda}.$$

By (12), we have

$$\begin{aligned} E[\tilde{T}_i] \le{}& \frac{n}{(n-i)\left(1+\sqrt{\frac{n}{n-i}}\sqrt{\frac{\lambda}{e\mu}}\right)} + 1 \\ &+ \frac{2e\mu}{\lambda}\left(\ln\frac{n}{(n-i)\left(1+\sqrt{\frac{n}{n-i}}\sqrt{\frac{\lambda}{e\mu}}\right)} + 2\right) \\ &+ \frac{e\mu}{\lambda}\left(1+\sqrt{\frac{n}{n-i}}\sqrt{1+\frac{\lambda}{e\mu}}\right). \end{aligned} \tag{13}$$

Let T_2 be the number of iterations until the $(\mu+\lambda)$ EA finds an individual with fitness greater than $\lfloor n - \frac{n}{\mu(2+\lambda/e)}\rfloor$ for the first time if it already has an individual with fitness greater than $\lceil n - \frac{n}{2+\lambda/(e\mu)}\rceil$ in the population. By Theorem 3 and by (13), we obtain

$$\begin{aligned} E[T_2] &\le \sum_{i=\lceil n-\frac{n}{2+\lambda/(e\mu)}\rceil+1}^{\lfloor n-\frac{n}{\mu(2+\lambda/e)}\rfloor} E[\tilde{T}_i] \\ &\le n \sum_{i=\lceil n-\frac{n}{2+\lambda/(e\mu)}\rceil+1}^{\lfloor n-\frac{n}{\mu(2+\lambda/e)}\rfloor} \frac{1}{(n-i)\left(1+\sqrt{\frac{n}{n-i}}\sqrt{\frac{\lambda}{e\mu}}\right)} \\ &\quad + \frac{e\mu}{\lambda}\sum_{i=\lceil n-\frac{n}{2+\lambda/(e\mu)}\rceil+1}^{\lfloor n-\frac{n}{\mu(2+\lambda/e)}\rfloor}\left(1+\sqrt{\frac{n}{n-i}}\sqrt{1+\frac{\lambda}{e\mu}}\right) \\ &\quad + \frac{2e\mu}{\lambda}\sum_{i=\lceil n-\frac{n}{2+\lambda/(e\mu)}\rceil+1}^{\lfloor n-\frac{n}{\mu(2+\lambda/e)}\rfloor}\ln\left(\frac{n}{(n-i)\Delta_i}\right) + \frac{2e\mu}{\lambda}2n + n. \end{aligned} \tag{14}$$

We regard three sums in (14) separately. First, by the estimate $\sum_{i=1}^{n} 1/\sqrt{i} \leq 1 + \int_1^n (1/\sqrt{x})dx < 2\sqrt{n}$, we obtain

$$\begin{aligned}
&\sum_{i=\lceil n-\frac{n}{2+\lambda/(e\mu)}\rceil+1}^{\lfloor n-\frac{n}{\mu(2+\lambda/e)}\rfloor} \frac{1}{(n-i)\left(1+\sqrt{\frac{n}{n-i}}\sqrt{\frac{\lambda}{e\mu}}\right)} \\
&\qquad \leq \sum_{i=\lceil n-\frac{n}{2+\lambda/(e\mu)}\rceil+1}^{\lfloor n-\frac{n}{\mu(2+\lambda/e)}\rfloor} \frac{1}{(n-i)\sqrt{\frac{n}{n-i}}\sqrt{\frac{\lambda}{e\mu}}} \\
&\qquad = \sqrt{\frac{e\mu}{\lambda n}} \sum_{i=\lceil n-\frac{n}{2+\lambda/(e\mu)}\rceil+1}^{\lfloor n-\frac{n}{\mu(2+\lambda/e)}\rfloor} \frac{1}{\sqrt{n-i}} \\
&\qquad \leq \sqrt{\frac{e\mu}{\lambda n}} \cdot 2\sqrt{\frac{n}{2+\lambda/(e\mu)}} \leq 2\sqrt{\frac{e\mu}{\lambda n}}\sqrt{\frac{n}{\lambda/(e\mu)}} = 2e\frac{\mu}{\lambda}.
\end{aligned} \tag{15}$$

To analyze the second sum we also use the estimate $\frac{1+t}{2+t} < 1$ valid for all $t \in [0, +\infty)$. We obtain

$$\begin{aligned}
&\sum_{i=\lceil n-\frac{n}{2+\lambda/(e\mu)}\rceil+1}^{\lfloor n-\frac{n}{\mu(2+\lambda/e)}\rfloor} \left(1+\sqrt{\frac{n}{n-i}}\sqrt{1+\frac{\lambda}{e\mu}}\right) \\
&\qquad \leq n + \sqrt{n\left(1+\frac{\lambda}{e\mu}\right)} \sum_{i=\lceil n-\frac{n}{2+\lambda/(e\mu)}\rceil+1}^{\lfloor n-\frac{n}{\mu(2+\lambda/e)}\rfloor} \frac{1}{\sqrt{n-i}} \\
&\qquad \leq n + \sqrt{n\left(1+\frac{\lambda}{e\mu}\right)} \cdot 2\sqrt{\frac{n}{2+\lambda/(e\mu)}} \\
&\qquad = n + 2n\sqrt{\frac{1+\lambda/(e\mu)}{2+\lambda/(e\mu)}} \leq 3n.
\end{aligned} \tag{16}$$

For the last sum we use the logarithmic version of Stirling's formula, that is, $\ln(n!) = n\ln(n) - n + O(\log(n))$ (see, e.g. [120] or [30, Theorem 1.4.10]), and the estimate $\frac{\ln(2+t)+2}{2+t} \leq 2$ for all $t \in [0, +\infty)$. We obtain

$$
\begin{aligned}
&\sum_{i=\lceil n-\frac{n}{2+\lambda/(e\mu)}\rceil+1}^{\lfloor n-\frac{n}{\mu(2+\lambda/e)}\rfloor} \ln\left(\frac{n}{(n-i)\Delta_i}\right) \\
&\qquad \le \sum_{i=\lceil n-\frac{n}{2+\lambda/(e\mu)}\rceil+1}^{\lfloor n-\frac{n}{\mu(2+\lambda/e)}\rfloor} \ln\left(\frac{n}{(n-i)}\right) \\
&\qquad \le \left\lceil \frac{n}{2+\lambda/(e\mu)} \right\rceil \ln(n) - \ln\left(\prod_{i=\lceil n-\frac{n}{2+\lambda/(e\mu)}\rceil}^{n} (n-i) \right) \\
&\qquad \le \left\lceil \frac{n}{2+\lambda/(e\mu)} \right\rceil \ln(n) - \ln\left(\left\lceil \frac{n}{2+\lambda/(e\mu)} \right\rceil ! \right) \\
&\qquad = \left\lceil \frac{n}{2+\lambda/(e\mu)} \right\rceil \left(\ln(n) - \ln\left\lceil \frac{n}{2+\lambda/(e\mu)} \right\rceil + 1 \right) \\
&\qquad + O\left(\log\left\lceil \frac{n}{2+\lambda/(e\mu)} \right\rceil \right) \\
&\qquad \le \frac{n}{2+\lambda/(e\mu)} \left(\ln\left(2+\frac{\lambda}{e\mu}\right) + 2 \right) + o(n) \\
&\qquad \le 2n + o(n).
\end{aligned}
\tag{17}
$$

Finally, by putting (15), (16) and (17) into (14) we obtain

$$
\begin{aligned}
E[T_2] &\le n \cdot 2e\frac{\mu}{\lambda} + \frac{e\mu}{\lambda} \cdot 3n + \frac{2e\mu}{\lambda}(2n + o(n)) + \frac{4e\mu n}{\lambda} + n \\
&= \frac{13e\mu n}{\lambda} + n + o\Big(\frac{\mu n}{\lambda}\Big) = O\Big(\frac{\mu n}{\lambda} + n\Big).
\end{aligned}
$$

For $n-1 \ge i > \lfloor n - \frac{n}{\mu(2+\lambda/e)} \rfloor$ we define $\mu_0(i) := \mu$. Note that this case can only appear when $\frac{n}{\mu(2+\lambda/e)} \ge 1$ and thus $\mu \le \frac{n}{(2+\lambda/e)} = O(n/\lambda)$. By Corollary 1 the expected waiting time for a fitness gain is at most

$$
E[\tilde{T}_i] \le \mu + \frac{2e\mu}{\lambda}(\ln(\mu)+1) + \frac{en}{\lambda(n-i)}.
$$

Let T_3 be the number of iterations until the $(\mu+\lambda)$ EA finds the optimum starting from the moment when it has an individual with fitness greater than $\lfloor n - \frac{n}{\mu(2+\lambda/e)} \rfloor$ in the population.

Then by Theorem 3 we have

$$\begin{aligned}
E[T_3] &\leq \sum_{i=\lfloor n-\frac{n}{\mu(2+\lambda/e)}\rfloor+1}^{n-1} E[\tilde{T}_i] \\
&\leq \sum_{i=\lfloor n-\frac{n}{\mu(2+\lambda/e)}\rfloor+1}^{n-1} \left(\mu + \frac{2e\mu}{\lambda}(\ln(\mu)+1) + \frac{en}{\lambda(n-i)}\right) \\
&\leq \mu\frac{n}{\mu(2+\lambda/e)} + \frac{2e\mu}{\lambda}(\ln(\mu)+1)\frac{n}{\mu(2+\lambda/e)} \\
&\quad + \frac{en}{\lambda}\left(\ln\frac{n}{\mu(2+\lambda/e)}+1\right) \\
&= O\Big(\frac{n}{\lambda}\Big) + O\left(\frac{n\log\mu}{\lambda^2}\right) + O\left(\frac{n\log n}{\lambda}\right) \\
&= O\left(\frac{n\log\mu}{\lambda^2}\right) + O\left(\frac{n\log n}{\lambda}\right) = O\Big(\frac{\mu n}{\lambda}\Big) + O\left(\frac{n\log n}{\lambda}\right).
\end{aligned}$$

Summing the expected runtimes for all optimization stages, we obtain the upper bound for the expected total runtime.

$$\begin{aligned}
E[T] &\leq E[T_1] + E[T_2] + E[T_3] \\
&= O\Big(\frac{\mu n}{\lambda} + n\Big) + O\Big(\frac{\mu n}{\lambda} + n\Big) + O\left(\frac{\mu n}{\lambda} + \frac{n\log n}{\lambda}\right) \\
&= O\left(\frac{n\log n}{\lambda} + \frac{\mu n}{\lambda} + n\right).
\end{aligned}$$

□

Upper Bound with Large λ

In this section we consider the case when $\frac{\lambda}{\mu} > e^e$. Due to the large number of offspring the algorithm performs significantly better in this case. The first reason of this speed-up is that the algorithm can now gain several fitness levels in one iteration with high probability when the current-best fitness is small. The second reason is the faster increase of the number of best individuals, see Corollary 2.

These two observations allow us to prove the following upper bound on the runtime.

Theorem 9. *If $\frac{\lambda}{\mu} \geq e^e$ then the expected number of iterations for the $(\mu+\lambda)$ EA to optimize the* ONEMAX *problem is*

$$O\left(\frac{n\log\log\frac{\lambda}{\mu}}{\log\frac{\lambda}{\mu}} + \frac{n\log n}{\lambda}\right).$$

Note that the bound given in Theorem 9 is asymptotically the same as the bound given in Theorem 8 when $\frac{\lambda}{\mu} = \Theta(1)$. The difference between the two bounds becomes asymptotically significant only when $\frac{\lambda}{\mu} = \omega(1)$. Therefore it does not matter which constant we choose to distinguish the fast regime of the algorithm. The main purpose of the choice of e^e as a border value is to simplify the proofs and to improve their readability. However without proof we note that all arguments used in this section hold also for the smaller values of $\frac{\lambda}{\mu}$ which are greater than $2e$.

To prove Theorem 9 we split the optimization process into four phases. Each phase corresponds to some range of the best fitness values, and the phase transition occurs at fitness values $n - \frac{n}{\ln \frac{\lambda}{\mu}}$, $n - \frac{\mu n}{\lambda}$ and $n - \frac{n}{\lambda}$. In each phase the $(\mu+\lambda)$ EA has a specific behavior, so we analyze each phase separately in the following four lemmas.

During the first phase, while the fitness of the best individual is below $n - \frac{n}{\ln \frac{\lambda}{\mu}}$, regardless of the number of best individuals, with constant probability we generate an offspring increasing the best fitness in the population by at least $\gamma := \lfloor \frac{\ln \frac{\lambda}{\mu}}{2 \ln \ln \frac{\lambda}{\mu}} \rfloor$. So we need not more than an expected number of $O(\frac{n}{\gamma})$ iterations to finish the first phase.

Let R_1 be the runtime of the $(\mu+\lambda)$ EA until it finds an individual with fitness at least $n - \frac{n}{\ln \frac{\lambda}{\mu}}$, in other words, the duration of the first phase. We prove the following upper bound on the expected value of R_1.

Lemma 35 (Phase 1). *If $\frac{\lambda}{\mu} \geq e^e$, then we have*

$$E[R_1] = O\left(\frac{n \log \log \frac{\lambda}{\mu}}{\log \frac{\lambda}{\mu}}\right).$$

Proof. To use Theorem 3, we split the space of populations S into levels $A_1, \dots A_m$, where

$$m := \left\lceil \frac{\lfloor n - \frac{n}{\ln \frac{\lambda}{\mu}} \rfloor}{\gamma} \right\rceil + 1.$$

If $k < m$, then the populations of level A_k have the fitness of the best individual in $[(k-1)\gamma..k\gamma-1]$ (but less than $n - \frac{n}{\ln \frac{\lambda}{\mu}}$). The level A_m consists of all populations containing an individual of fitness at least $n - \frac{n}{\ln \frac{\lambda}{\mu}}$.

To show that we have a constant probability to leave any level, we consider the probability that a particular offspring has a fitness exceeding the current best fitness i by at least γ. This is at least the probability to choose one of the best individuals and to flip exactly γ zero-bits in it and not to flip the other $n - \gamma$ bits, namely

$$\binom{n-i}{\gamma} \frac{j}{\mu n^\gamma} \left(1 - \frac{1}{n}\right)^{n-\gamma} \geq \frac{j}{e\mu} \left(\frac{n-i}{n\gamma}\right)^\gamma =: p_\gamma(i).$$

The probability to increase the best fitness by at least γ with one of λ offspring is at least $1 - (1 - p_\gamma(i))^\lambda$. Thus, by Lemma 21, the expected number of iterations for this to happen is not larger than

$$\frac{1}{1 - (1 - p_\gamma(i))^\lambda} \leq 1 + e\frac{\mu}{\lambda} \left(\frac{n\gamma}{n-i}\right)^\gamma.$$

Since $\frac{\lambda}{\mu} \geq e^e$, we have $\gamma = \lfloor \frac{\ln \frac{\lambda}{\mu}}{2 \ln \ln \frac{\lambda}{\mu}} \rfloor \geq \lfloor \frac{e}{2} \rfloor = 1$. Using this and the estimate $\frac{n}{n-i} \leq \ln \frac{\lambda}{\mu}$ valid during this phase, we compute

$$\left(\frac{n\gamma}{n-i}\right)^{\gamma} \leq \exp\left(\gamma \ln\left(\gamma \ln \frac{\lambda}{\mu}\right)\right) \leq \exp\left(\frac{\ln \frac{\lambda}{\mu}}{2 \ln \ln \frac{\lambda}{\mu}} \ln\left(\frac{\ln^2 \frac{\lambda}{\mu}}{2 \ln \ln \frac{\lambda}{\mu}}\right)\right)$$
$$\leq \exp\left(\frac{\ln \frac{\lambda}{\mu}}{2 \ln \ln \frac{\lambda}{\mu}} 2 \ln \ln \frac{\lambda}{\mu}\right) = \exp\left(\ln \frac{\lambda}{\mu}\right) = \frac{\lambda}{\mu}.$$

Therefore, the expected time to increase the fitness by γ (and thus to leave level A_k for any $k < m$) is at most $1 + e$. Summing over the levels $A_1, \ldots, A_{m-1}$, by Theorem 3 we have

$$E[R_1] \leq \sum_{k=1}^{m-1} (1+e) < (1+e)m < (1+e)\frac{n}{\gamma} = O\left(\frac{n \log \log \frac{\lambda}{\mu}}{\log \frac{\lambda}{\mu}}\right).$$

□

Having found an individual with fitness at least $n - \frac{n}{\ln \frac{\lambda}{\mu}}$, we enter the second phase. Due to the elitist selection, the minimum fitness in the population does not decrease, so there is no risk of a fall-back into the first phase.

In the second phase, due to the smaller distance from the optimum, fitness gains by more than a constant are too rare to be exploited profitably. However, even when we only have one best individual in the population, the probability to create at least one better individual in one iteration will still be constant. Consequently, we do not need to analyze how the the number of best individuals grows. This phase ends when the best fitness in the population is $n - \frac{\mu n}{\lambda}$ or more.

Let R_2 be the runtime of the $(\mu + \lambda)$ EA until it finds an individual with fitness at least $n - \frac{\mu n}{\lambda}$ starting from the moment when it has an individual with fitness at least $n - \frac{n}{\ln \frac{\lambda}{\mu}}$ in the population. In other words, R_2 is the duration of the second phase.

Lemma 36 (Phase 2)**.** *If $\frac{\lambda}{\mu} \geq e^e$, then we have*

$$E[R_2] = O\left(\frac{n}{\log \frac{\lambda}{\mu}}\right).$$

Proof. For

$$i \in \left[\left\lceil n - \frac{n}{\ln \frac{\lambda}{\mu}} \right\rceil .. \left\lceil n - \frac{\mu n}{\lambda} \right\rceil - 1\right],$$

the level B_i is defined as the set of all populations in which the best individuals have fitness i. For $i = \lceil n - \frac{\mu n}{\lambda} \rceil$ let the level B_i consist of all populations with best fitness at least i.

By Corollary 2 and defining $\mu_0(i) \coloneqq 1$ for all i, we have

$$E[\tilde{T}_i] \leq \frac{e\mu n}{\lambda(n-i)} + 5 \leq e + 5,$$

where the last estimate follows from $i \le n - \frac{\mu n}{\lambda}$. Therefore, by Theorem 3

$$E[R_2] \le \sum_{i=\lceil n-n/\ln\frac{\lambda}{\mu}\rceil}^{\lceil n-n\mu/\lambda\rceil-1} E[\tilde{T}_i] \le (5+e)\frac{n}{\ln\frac{\lambda}{\mu}} = O\left(\frac{n}{\log\frac{\lambda}{\mu}}\right).$$

□

After completion of the second phase, generating a strictly better individual is so difficult that it pays off (in the analysis) to wait for more than one best individual in the population. More precisely, depending on the current best fitness i we define a number $\mu_0(i)$ and compute the time to reach $\mu_0(i)$ best individuals and argue that the expected time to generate a strict improvement when at least $\mu_0(i)$ best individuals are in the population is only constant. Since, as discussed in Lemma 33, the number of the best individuals in the population roughly increases by a factor $(1+\frac{\lambda}{2e\mu})$ in each iteration, the algorithm obtains $\mu_0(i)$ individuals reasonably fast.

Let R_3 be the runtime of the $(\mu+\lambda)$ EA until it finds an individual with fitness at least $n-\frac{n}{\lambda}$, the end of the third phase, starting from the moment when it has an individual with fitness at least $n-\frac{\mu n}{\lambda}$ in the population.

Lemma 37 (Phase 3)**.** *If $\frac{\lambda}{\mu} \ge e^e$, then we have*

$$E[R_3] = O\Big(\frac{\mu n}{\lambda}\Big).$$

Proof. During this phase the best fitness i in the population satisfies

$$n - \frac{\mu n}{\lambda} \le i < n - \frac{n}{\lambda},$$

which implies

$$\frac{\lambda}{\mu} \le \frac{n}{n-i} < \lambda. \tag{18}$$

For these values of i we define $\mu_0(i) := \lceil \frac{n\mu}{(n-i)\lambda} \rceil$. Note that $\mu_0(i) \in [1..\mu]$.

For

$$i \in \left[\lceil n - \frac{\mu n}{\lambda}\rceil .. \lceil n - \frac{n}{\lambda}\rceil - 1\right],$$

level C_i is defined as a set of all populations in which the best individuals have fitness i. For $i = \lceil n - \frac{n}{\lambda}\rceil$ let the level C_i consist of all populations with best fitness at least i.

By Corollary 2 and by the definition of $\mu_0(i)$ we have

$$\begin{aligned} E[\tilde{T}_i] &\le 4\frac{\ln \mu_0(i)}{\ln\frac{\lambda}{2e\mu}} + \frac{e\mu n}{\lambda(n-i)\mu_0(i)} + 5 \\ &\le \frac{4}{\ln\frac{\lambda}{2e\mu}}\left(\ln\frac{n\mu}{(n-i)\lambda} + 1\right) + e + 5. \end{aligned}$$

By Theorem 3 we obtain

$$E[R_3] \le \sum_{i=\lceil n-n\mu/\lambda\rceil}^{\lceil n-n/\lambda\rceil-1} \tilde{T}_i$$

$$\leq \sum_{i=\lceil n-n\mu/\lambda\rceil}^{\lceil n-n/\lambda\rceil-1} \left(\frac{4}{\ln \frac{\lambda}{2e\mu}} \left(\ln \frac{n\mu}{(n-i)\lambda} + 1 \right) + e + 5 \right)$$
$$\leq \frac{4}{\ln \frac{\lambda}{2e\mu}} \left(\frac{n\mu}{\lambda} + \sum_{i=\lceil n-n\mu/\lambda\rceil}^{\lceil n-n/\lambda\rceil-1} \ln \frac{n\mu}{(n-i)\lambda} \right) + \frac{n\mu}{\lambda}(e+5).$$

We estimate $\sum_{i=\lceil n-n\mu/\lambda\rceil}^{\lceil n-n/\lambda\rceil-1} \ln \frac{n\mu}{(n-i)\lambda}$ using Stirling's formula as in (17). We also notice that this phase occurs only when $\frac{n\mu}{\lambda} > 1$, thus we have $(\ln \frac{n\mu}{\lambda} - \ln\lfloor \frac{n\mu}{\lambda} \rfloor) \leq 1$. Hence, we obtain.

$$\begin{aligned}
\sum_{i=\lceil n-n\mu/\lambda\rceil}^{\lceil n-n/\lambda\rceil-1} \ln \frac{n\mu}{(n-i)\lambda} &\leq \sum_{i=1}^{\lfloor n\mu/\lambda\rfloor} \ln \frac{n\mu}{i\lambda} \\
&= \left\lfloor \frac{n\mu}{\lambda} \right\rfloor \ln \frac{n\mu}{\lambda} - \left\lfloor \frac{n\mu}{\lambda} \right\rfloor \ln \left\lfloor \frac{n\mu}{\lambda} \right\rfloor \\
&+ \left\lfloor \frac{n\mu}{\lambda} \right\rfloor + O\Big(\log \left\lfloor \frac{n\mu}{\lambda} \right\rfloor \Big) \\
&= \left\lfloor \frac{n\mu}{\lambda} \right\rfloor \left(\ln \frac{n\mu}{\lambda} - \ln \left\lfloor \frac{n\mu}{\lambda} \right\rfloor + 1 \right) + o\Big(\frac{\mu n}{\lambda} \Big) \\
&\leq 2\frac{n\mu}{\lambda} + o\Big(\frac{\mu n}{\lambda} \Big).
\end{aligned}$$

Therefore,

$$E[R_3] \leq (5+e)\frac{\mu n}{\lambda} + 4\frac{2\frac{n\mu}{\lambda} + o\left(\frac{\mu n}{\lambda}\right) + \frac{n\mu}{\lambda}}{\ln \frac{\lambda}{2e\mu}} = O\Big(\frac{\mu n}{\lambda} \Big).$$

□

When the algorithm is closer to the optimum than in the third phase, then we cannot expect to have a constant probability for a strict fitness improvement even when the whole population consists of individuals of best fitness. In this forth and last phase, we thus always wait (in the analysis) until the population only contains best individuals and then estimate the expected time for an improvement. We denote by R_4 the runtime until the algorithm finds the optimum if it already has an individual with fitness at least $n - \frac{n}{\lambda}$ in the population.

Lemma 38 (Phase 4). *If* $\frac{\lambda}{\mu} \geq e^e$ *then*

$$E[R_4] = O\left(\frac{n \log n}{\lambda} \right).$$

Proof. For

$$i \in \left[\left\lceil n - \frac{n}{\lambda} \right\rceil .. n-1 \right]$$

we define level D_i as a set of all populations in which the best individuals have fitness i. We also define $\mu_0(i) = \mu$ for these values of i.

By Corollary 2 we have

$$E[\tilde{T}_i] \leq 4\frac{\ln \mu}{\ln \frac{\lambda}{2e\mu}} + \frac{en}{\lambda(n-i)} + 5.$$

Therefore, by Theorem 3, we obtain

$$\begin{aligned} E[R_4] &\leq \sum_{i=\lceil n-\frac{n}{\lambda}\rceil}^{n-1} \left(\frac{4\ln \mu}{\ln \frac{\lambda}{2e\mu}} + \frac{en}{\lambda(n-i)} + 5 \right) \\ &\leq \frac{4n\ln \mu}{\lambda \ln \frac{\lambda}{2e\mu}} + \frac{en(\ln \frac{n}{\lambda}+1)}{\lambda} + \frac{5n}{\lambda} \\ &= O\left(\frac{n}{\log \frac{\lambda}{\mu}}\right) + O\left(\frac{n\log n}{\lambda}\right). \end{aligned}$$

□

Finally, we prove Theorem 9.

Proof (Theorem 9). Since we consider an elitist algorithm that cannot reduce the best fitness, by linearity of expectation and Lemmas 35 to 38 we have

$$E[T] \leq E[R_1] + E[R_2] + E[R_3] + E[R_4] = O\left(\frac{n\log\log \frac{\lambda}{\mu}}{\log \frac{\lambda}{\mu}} + \frac{n\log n}{\lambda}\right).$$

□

Comparison With Other Upper Bounds

We first note that our upper bound

$$O\left(\frac{n\log n}{\lambda} + \frac{n}{\lambda/\mu} + \frac{n\log^+\log^+(\lambda/\mu)}{\log^+(\lambda/\mu)}\right)$$

for the runtime of the $(\mu+\lambda)$ EA on OneMax subsumes the known bounds

$$O(n\log n + \mu n)$$

for the $(\mu+1)$ EA [138] and

$$O\left(\frac{n\log n}{\lambda} + \frac{n\log^+\log^+\lambda}{\log^+\lambda}\right)$$

for the $(1+\lambda)$ EA [49].

We are not aware of any previous result for the $(\mu+\lambda)$ EA for general values of μ and λ. We believe that a domination argument allows to transfer the results of Corus et al. [17] for the

(μ, λ) EA to the $(\mu + \lambda)$ EA. Since we prove in this work a bound that is at least as strong and stronger in some cases, we sketch this argument now, but do not give formal proofs.

We recall that the (μ, λ) EA differs from the $(\mu + \lambda)$ EA only in the selection mechanism, which disallows the (μ, λ) EA to select any parent individual into the next population. This imposes a constraint on the parameters requiring λ to be at least μ. For the case that $\lambda > (1+\delta)e\mu$, $\delta > 0$ a constant, and $\lambda \geq c \ln(n)$ with sufficiently large constant c, Corus et al. [17, Theorem 3] proved that the (μ, λ) EA within an expected number of $O(n)$ iterations finds the optimum of OneMax.

Since the $(\mu + \lambda)$ EA uses elitist selection, we conjecture that the fitness values of its population always stochastically dominate those of the population of the (μ, λ) EA. More precisely, for a run of the $(\mu + \lambda)$ EA let us for $i \in [1..\mu]$ and $t \in \mathbb{N}$ denote by f_{it} the fitness of the i-th individual in the parent population after iteration t, where we assume that the individuals are sorted by decreasing fitness. Let us denote by f'_{it} the same for the (μ, λ) EA. Then for all i and t, the random variable f_{it} stochastically dominates f'_{it}. Presumably, this can be shown via coupling in a similar fashion as in the proof of Theorem 23 in [26]. Thus the upper bound given by Corus et al. is also valid for the $(\mu + \lambda)$ EA. For the case $\lambda > (1 + \delta)e\mu$ and $\lambda = \Omega(\log n)$ regarded by Corus et al., our bound becomes

$$O\left(n\left(\frac{\log(n)}{\lambda} + \frac{\mu}{\lambda} + \frac{\log^+ \log(\lambda/\mu)}{\log(\lambda/\mu)}\right)\right),$$

which is of an asymptotically slightly smaller order than that of [17] when $\lambda = \omega(\mu + \log(n))$.

2.2.2 Lower Bounds

In this section, we show the lower bounds corresponding to the upper bounds we proved in the previous section. They in particular imply the lower bounds for the $(\mu + 1)$ EA given in [138] and the $(1 + \lambda)$ EA given in [49]. Hence our proof method is a unified approach to both these algorithms as well. The arguments we use do not consider selection phase at all, thus they hold also for all functions with a unique optimum and for other selection mechanisms, including the (μ, λ) EA.

To prove the lower bound we use the complete tree technique. We start with observation thatsince there is no selection in the complete tree, the vertex labels simply arise from repeated mutation. More precisely, a vertex in distance ℓ from the root has a label that is obtained from ℓ times applying mutation to the root label. This elementary observation allows to estimate the probability that a node label is equal to some target string.

Lemma 39. *Consider a complete tree with root label $c(v_0) = x_0$. Let $x^* \in \{0,1\}^n$ with $H(x^*, x_0) \geq n/4$ (where H is the Hamming distance). Let x be the node label of a node in distance ℓ from v_0. Then*

$$\Pr[x = x^*] \leq \min\left\{1, \left(\frac{\ell}{n-1}\right)^{n/4}\right\} =: p(\ell, n).$$

Proof. The probability that $x = x^*$ is at most the probability that each of the $H(x_0, x^*)$ bits in which x_0 and x^* differ was flipped in at least one of the ℓ applications of the mutation operator

which generated x from x_0. For one particular position the probability that this position was involved in one of ℓ mutations is $1-(1-\frac{1}{n})^\ell$. For $H(x_0,x^*)$ positions the probability that all of them were involved in one of ℓ mutations is

$$\left(1-\left(1-\frac{1}{n}\right)^\ell\right)^{H(x_0,x^*)} \leq \left(1-\exp\left(-\frac{\ell}{n-1}\right)\right)^{\frac{n}{4}} \leq \left(\frac{\ell}{n-1}\right)^{\frac{n}{4}},$$

where we used the estimates $(1-1/n)^{(n-1)r} \geq e^{-r}$ valid for all $n \geq 1$ and any positive $r \in \mathbb{R}$, and $e^{-r} \geq 1-r$ valid for all $r \in \mathbb{R}$. □

We are now ready to prove our lower bound. Since the proof is valid not only for the ONEMAX function, but for any pseudo-Boolean function with a unique optimum, we formulate the result for such functions. We show extensions to many functions with multiple optima in the following section.

Theorem 10. *If μ is polynomial in n, then the $(\mu+\lambda)$ EA with any type of selection of the new parent population (including only selecting from the offspring population) needs an expected number of*

$$\Omega\left(\frac{n\log n}{\lambda}+\frac{\mu n}{\lambda}\right)$$

iterations to optimize any pseudo-Boolean function with a unique optimum.

If further $\frac{\lambda}{\mu} \geq e^e$, then the stronger bound

$$\Omega\left(\frac{n\log n}{\lambda}+\frac{n\log\log\frac{\lambda}{\mu}}{\log\frac{\lambda}{\mu}}\right)$$

holds.

Proof. Without any loss of generality in this proof we assume that the function optimized by the algorithm has an optimum in $x^*=(1,\ldots,1)$.

In our proofs we use the following tool. To prove that the expected runtime of the algorithm is $\Omega(f(n))$ for some function $f(n)$, it is enough to prove that the probability that the runtime is less than $f(n)$ is less than some constant $\gamma<1$, since in this case the expected runtime is not less than $(1-\gamma)f(n)$.

We first note that the bound $\Omega(\frac{n\log n}{\lambda})$ is easy to prove for the ONEMAX function. A short, but deep argument for this bound is that the $(\mu+\lambda)$ EA is an unary unbiased black-box complexity algorithm in the sense of Lehre and Witt [97]. Any such algorithm needs an expected number of $\Omega(n\log n)$ [97] or, more precisely, of at least $n\ln(n)-O(n)$ [40] fitness evaluations to find the optimum of the ONEMAX function.

However, we prove the lower bounds for any function with a unique optimum, so we use an elementary argument essentially identical to the one of [138] as follows. The lower bound $\Omega(\frac{n\log n}{\lambda})$ needs to be shown only in the case $\mu \leq c\log n$, where c is an arbitrarily small constant. For any bit position we have the probability q_1 that all individuals in the initial population have a zero-bit in that position that is calculated as

$$q_1=\left(\frac{1}{2}\right)^\mu \geq \left(\frac{1}{2}\right)^{c\log(n)} = \exp(c\log(n)\log(1/2)) = n^{-c\log(2)}.$$

Thus, the expected number z of the bit positions such that all individuals in the initial population have a zero-bit in that position is

$$E[z] = \sum_{i=1}^{n} q_1 = n^{1-c\log(2)}.$$

We call such positions *initially wrong positions*. Since the bit values in each bit position and each initial individual are independent, each position is initially wrong or not independently on other positions. Hence, by Chernoff bounds (see, e.g., Theorem 1.10.5 in [30]) the probability q_2 that we have at least $E[z]/2 = n^{1-c\log(2)}/2$ such bit positions is calculated as

$$q_2 = \Pr\left[z \geq (1-\delta)E[z]\right] \geq 1 - \exp\left(-\frac{\delta^2 E[z]}{2}\right) = 1 - \exp\left(-\frac{n^{1-c\log(2)}}{8}\right).$$

Now we are ready to show that the algorithm does not flip at least one of the bits in the initially wrong positions in $t := \lfloor \frac{\alpha(n-1)\log(n)}{\lambda} \rfloor$ iterations, where α is a constant that will be defined later, with a high (at least $1-o(1)$) probability. We calculate the probability q_3 that one particular bit is flipped at least once in t iterations (or in λt mutations) as

$$\begin{aligned} q_3 &= 1 - \left(1 - \frac{1}{n}\right)^{\lambda t} = 1 - \left(1 - \frac{1}{n}\right)^{(n-1)\frac{\lambda t}{(n-1)}} \\ &\leq 1 - \exp\left(-\frac{t\lambda}{(n-1)}\right) \leq 1 - e^{-\alpha\log(n)} = 1 - n^{-\alpha}. \end{aligned}$$

If we have at least $n^{1-c\log(2)}/2$ initially wrong positions, then the probability q_4 that all of them are flipped at least once in t iterations is

$$\begin{aligned} q_4 &= q_3^{\frac{n^{1-c\log(2)}}{2}} \leq (1-n^{-\alpha})^{\frac{n^{1-c\log(2)}}{2}} \\ &= (1-n^{-\alpha})^{n^{\alpha}\cdot\frac{n^{1-c\log(2)-\alpha}}{2}} \leq \exp\left(-\frac{n^{1-c\log(2)-\alpha}}{2}\right) \end{aligned}$$

Thus we have the probability q_5 that at least one of the initially wrong bits is not flipped (and thus, the optimum is not found) in $t = \Theta(\frac{n\log(n)}{\lambda})$ iterations at least

$$\begin{aligned} q_5 &\geq q_2(1-q_4) \\ &\geq \left(1 - \exp\left(-\frac{n^{1-c\log(2)}}{8}\right)\right)\left(1 - \exp\left(-\frac{n^{1-c\log(2)-\alpha}}{2}\right)\right) \\ &\geq 1 - 2\exp\left(-\frac{n^{1-c\log 2-\alpha}}{8}\right). \end{aligned} \tag{19}$$

Hence, if α and c satisfy $c\log(2) + \alpha < 1$, (e.g., $\alpha := \frac{1}{2}$ and $c := \frac{1}{2}$) then the expected runtime of the algorithm is $\Omega(\frac{n\log(n)}{\lambda})$.

To prove the remaining two bounds, we argue as follows. Again using a simple Chernoff bound argument, we first observe that the probability q_6 that the number of zero-bits y in the one particular individual in the initial population is less than $n/4$, is estimated as

$$q_6 = \Pr\left[y \leq \frac{E[y]}{2}\right] = \Pr\left[y \leq (1-1/2)E[y]\right] \leq \exp\left(-\frac{n}{16}\right).$$

Hence, all μ individuals of the initial population have a Hamming distance of at least $n/4$ from the optimum x^* with probability

$$q_7 = (1-q_6)^\mu \geq \left(1-\exp\left(-\frac{n}{16}\right)\right)^\mu \geq \exp\left(-\frac{\mu}{e^{\frac{n}{16}}-1}\right)$$

Since μ is polynomial in n, we have $\frac{\mu}{e^{\frac{n}{16}}-1} = o(1)$ and therefore, $q_7 = 1-o(1)$. Further in this proof we assume that all initial individuals have at least $n/4$ zero-bits. Therefore, the probability that an individual which is a label of vertex in distance ℓ from the root of a complete tree is the optimum is at most $p(\ell,n)$ as shown in Lemma 39.

We now use the complete trees technique and Theorem 6 to prove the remaining lower bounds. Let first $t := \lfloor \mu n/8e\lambda \rfloor$. Using the inequality $\binom{t}{\ell} \leq (et/\ell)^\ell$ that follows from Stirling's formula, we estimate the summand $s(\ell) := \binom{t}{\ell}(\frac{\lambda}{\mu})^\ell p(\ell,n)$ of q_{opt} for every $\ell \in [0..t]$.

- By Lemma 39 we have $p(\ell,n) \leq 1$. Thus, if $\ell \geq n/4$, we estimate

$$s(\ell) = \binom{t}{\ell}\left(\frac{\lambda}{\mu}\right)^\ell p(\ell,n) \leq \left(\frac{et\lambda}{\ell\mu}\right)^\ell \leq \left(\frac{n}{8\ell}\right)^\ell \leq (1/2)^\ell \leq (1/2)^{n/4}.$$

- By Lemma 39 we have $p(\ell,n) \leq (\ell/(n-1))^{n/4}$. Hence, if $n/4 \geq \ell > 0$, we estimate

$$\begin{aligned} s(\ell) = \binom{t}{\ell}\left(\frac{\lambda}{\mu}\right)^\ell p(\ell,n) &\leq \left(\frac{et\lambda}{\ell\mu}\right)^\ell \left(\frac{\ell}{n-1}\right)^{n/4} \leq \left(\frac{n}{8\ell}\right)^\ell \left(\frac{\ell}{n-1}\right)^{n/4} \\ &\leq \left(\frac{n}{4\ell}\right)^\ell \left(\frac{\ell}{n-1}\right)^{n/4} \leq \left(\frac{n}{4\ell}\cdot\frac{\ell}{n-1}\right)^{n/4} \leq (1/2)^{n/4}. \end{aligned}$$

- Finally, for $\ell = 0$ we have $p(\ell,n) = 0$ and thus $s(\ell) = 0$.

Consequently, the optimum is found in less than t iterations if either there is an individual with less than $n/4$ zero-bits in the initial population, or with an exponentially small probability otherwise. Therefore, the probability q_8 of finding the optimum in less than t iterations is bounded as

$$\begin{aligned} q_8 &\leq (1-q_7) + q_7\mu\sum_{\ell=0}^{t} s(\ell) \\ &\leq \left(1-\exp\left(-\frac{\mu}{e^{\frac{n}{16}}-1}\right)\right) + \mu\sum_{\ell=1}^{t}(1/2)^{n/4} \qquad (20) \\ &\leq \frac{\mu}{e^{\frac{n}{16}}-1} + \frac{\mu^2 n}{8e\lambda}(1/2)^{n/4} = o(1), \end{aligned}$$

since we assumed μ to be at most polynomial in n.

We finish the proof by showing the lower bound $\Omega\left(\frac{n\log\log\frac{\lambda}{\mu}}{\log\frac{\lambda}{\mu}}\right)$ in case when $\frac{\lambda}{\mu} \geq e^e$. For this purpose let $t = \lfloor \frac{(e-2)n\ln\ln\frac{\lambda}{\mu}}{4(e+1)\ln\frac{\lambda}{\mu}} \rfloor$. Using the complete tree notation we show that the probability that the algorithm finds an optimum in less than t iterations is very small.

For all $\ell \in [0..t]$ consider $s(\ell)$. Using the inequality $\binom{t}{\ell} \leq (et/\ell)^\ell$ we estimate the upper bound for it as follows.

$$\begin{aligned} s(\ell) &= \binom{t}{\ell}\left(\frac{\lambda}{\mu}\right)^{\ell} p(\ell,n) \le \left(\frac{et\lambda}{\ell\mu}\right)^{\ell}\left(\frac{\ell}{n-1}\right)^{n/4} \\ &= \exp\left(\ell \ln\frac{et\lambda}{\ell\mu} + \frac{n}{4}\ln\frac{\ell}{n-1}\right). \end{aligned} \tag{21}$$

Consider precisely the argument of the exponential function from the last equality in (21). For this purpose define $f(\ell) := \ell \ln \frac{et\lambda}{\ell\mu} + \frac{n}{4}\ln\frac{\ell}{n-1}$. By considering the derivative of $f(\ell)$ on segment $[0,t]$ one can see that it is a monotonically increasing function. Since $t \ge \ell$ and $\frac{\lambda}{\mu} \ge e^e$, we have $\frac{et\lambda}{\ell\mu} \ge e^{e+1}$ and thus, $\ln\frac{et\lambda}{\ell\mu} \ge e+1$. Hence,

$$f'(\ell) = \ln\frac{et\lambda}{\ell\mu} - 1 + \frac{n}{4\ell} \ge e+1-1 > 0$$

Thus, $f(\ell)$ reaches its maximum when $\ell = t$. Therefore,

$$\begin{aligned} f(\ell) &\le f(t) \le t\ln\frac{e\lambda}{\mu} + \frac{n}{4}\ln\frac{t}{n-1} \\ &\le \frac{(e-2)n\ln\ln\frac{\lambda}{\mu}}{4(e+1)\ln\frac{\lambda}{\mu}}\left(\ln\frac{\lambda}{\mu}+1\right) + \frac{n}{4}\ln\frac{(e-2)n\ln\ln\frac{\lambda}{\mu}}{4(e+1)(n-1)\ln\frac{\lambda}{\mu}} \\ &= \frac{(e-2)}{4(e+1)}n\ln\ln\frac{\lambda}{\mu}\left(1+\frac{1}{\ln\frac{\lambda}{\mu}}\right) \\ &+ \frac{n}{4}\left(\ln\ln\ln\frac{\lambda}{\mu} - \ln\ln\frac{\lambda}{\mu} + \ln\frac{(e-2)n}{4(e+1)(n-1)}\right) \\ &\le \frac{n}{4}\ln\ln\frac{\lambda}{\mu}\left(\frac{(e-2)}{(e+1)}\left(1+\frac{1}{e}\right) + \frac{\ln\ln\ln\frac{\lambda}{\mu}}{\ln\ln\frac{\lambda}{\mu}} - 1 + \frac{\ln\frac{(e-2)n}{4(e+1)(n-1)}}{\ln\ln\frac{\lambda}{\mu}}\right). \end{aligned}$$

Notice that $\frac{\ln x}{x} \le \frac{1}{e}$ for all $x \ge 1$ and that $\ln\frac{(e-2)n}{4(e+1)(n-1)} < 0$ for all $n > 1$. Therefore we have

$$f(\ell) \le \frac{n}{4}\ln\ln\frac{\lambda}{\mu}\left(\frac{(e-2)}{(e+1)}\left(1+\frac{1}{e}\right) - \left(1-\frac{1}{e}\right)\right) = -\frac{n\ln\ln\frac{\lambda}{\mu}}{4e}.$$

Thus, by (21) we have

$$s(\ell) \le \exp\left(-\frac{n\ln\ln\frac{\lambda}{\mu}}{4e}\right) = \left(\ln\frac{\lambda}{\mu}\right)^{-n/4e}.$$

By Theorem 6 summing up $\mu s(\ell)$ for all $\ell \in [0..t]$ we obtain the following upper bound on the probability q_9 that the algorithm finds the optimum in less than $t = \Theta(\frac{n\log\log\frac{\lambda}{\mu}}{\log\frac{\lambda}{\mu}})$ iterations.

$$\begin{aligned} q_9 &\leq (1-q_7) + q_7\mu \sum_{\ell=0}^{t} s(\ell) \\ &\leq \frac{\mu}{e^{\frac{n}{16}}-1} + \mu \frac{(e-2)n \ln\ln\frac{\lambda}{\mu}}{4(e+1)\ln\frac{\lambda}{\mu}} \left(\ln\frac{\lambda}{\mu}\right)^{-n/4e}. \end{aligned} \tag{22}$$

Notice that q_9 is $o(1)$, since we assumed that μ is polynomial in n. Hence, the expected runtime of the algorithm is $\Omega(\frac{n\log\log\frac{\lambda}{\mu}}{\log\frac{\lambda}{\mu}})$ □

Comparison With Other Lower Bounds

Since all results involved are asymptotically tight, our lower bounds subsume the previous bounds for the $(\mu+1)$ EA and the $(1+\lambda)$ EA in the way as discussed for upper bounds.

For general values of μ and λ, the only result [116] we are aware of proves that for any μ and λ that are at most polynomial in n the runtime of the $(\mu+\lambda)$ EA on every pseudo-boolean function with a unique global optimum is

$$\Omega\left(\frac{n\log n}{\lambda} + \frac{\mu}{\lambda} + \frac{n\log\log n}{\log n}\right). \tag{23}$$

By comparing the three terms of this bound with the corresponding terms of our bound

$$\Omega\left(\frac{n\log n}{\lambda} + \frac{n}{\lambda/\mu} + \frac{n\log^+\log^+(\lambda/\mu)}{\log^+(\lambda/\mu)}\right),$$

we immediately see that our bound is asymptotically at least as large as the one in (23); note that for the third term, this follows trivially from the assumption that λ is polynomial in n and the fact that $x \mapsto \frac{\log\log(x)}{\log(x)}$ is decreasing for x sufficiently large.

There are two cases when our bound is asymptotically greater than (23).

Setting 1. Let $\frac{\lambda}{\mu} = O(1)$ and $\mu = \omega(\log(n))$. Then our bound is $\Omega(\frac{n\mu}{\lambda})$, which is at least $\Omega(n)$. On the other hand, (23) is

$$\frac{n\log n}{\lambda} + \frac{\mu}{\lambda} + \frac{n\log\log n}{\log n} = \frac{n\,o(\mu)}{\lambda} + \frac{\mu}{\lambda} + o(n) = o\left(\frac{n\mu}{\lambda}\right).$$

Setting 2. Let $\log\frac{\lambda}{\mu} = \omega(\log n)$. This implies that $\frac{\lambda}{\mu} = \omega(n)$ and thus

$$\log n = o\left(\frac{n\log\log n}{\log n}\right) = o\left(\frac{\frac{\lambda}{\mu}\log\log\frac{\lambda}{\mu}}{\log\frac{\lambda}{\mu}}\right).$$

Therefore, we have

$$\frac{n\log n}{\lambda} = o\left(\frac{n\log\log\frac{\lambda}{\mu}}{\mu\log\frac{\lambda}{\mu}}\right) = o\left(\frac{n\log\log\frac{\lambda}{\mu}}{\log\frac{\lambda}{\mu}}\right).$$

Hence, the lower bound given in Theorem 10 simplifies to $\Omega(\frac{n\log\log\frac{\lambda}{\mu}}{\log\frac{\lambda}{\mu}})$.

On the other hand, the bound (23) is of the asymptotically smaller order $o(\log n) + o(1) + O(\frac{n\log\log n}{\log n}) = O(\frac{n\log\log n}{\log n})$.

2.2.3 Extending the Lower Bounds to All Functions Having Not Excessively Many Global Optima

Since the family tree technique depends little on the particular function to be optimized, Witt [138] extended his lower bounds for ONEMAX to a much broader class of functions. He proved that the $(\mu+1)$ EA needs $\Omega(\mu n)$ iterations to find a global optimum of any function that satisfies one of the following conditions. (i) The function has at most $2^{o(n)}$ optima. (ii) All optima have at least $n/2+\varepsilon n$ one-bits or all optima have at least $n/2+\varepsilon n$ zero-bits, where $\varepsilon>0$ is an arbitrary constant.

In this section we extend our lower bounds of Section 2.2.2 to a wide class of functions as well. In particular, we show that Witt's results are valid for all functions with at most $2^{\beta n}$ optima, where β is some constant less than $\frac{1}{16\ln 2}$, regardless of the positions of the optima.

To reach our goal we exploit the fact that in Theorem 10 we proved very small values for the probabilities that the runtime is less than some threshold (see (19), (20) and (22)), while it would have been enough to prove that they are some constants less than one.

Theorem 11. *For any constant $\varepsilon>0$ there exists another constant $c>0$ such that if $\mu<c\ln n$, then for any n-dimensional pseudo-Boolean function with not more than $2^{n^{1-\varepsilon}}$ optima the $(\mu+\lambda)$ EA takes at least $\Omega(\frac{n\log n}{\lambda})$ iterations in expectation and with high probability to find an optimum.*

Proof. Let c be some arbitrary small positive constant and let $\mu<c\ln n$. By (19) the probability that the algorithm finds a particular optimum in less than $t:=\frac{\alpha n\log n}{\lambda}$ iterations (where α is some arbitrary constant) is

$$1-q_5\le 2\exp\left(-\frac{n^{1-c\ln 2-\alpha}}{8}\right).$$

If we have at most $2^{n^{1-\varepsilon}}$ optima, then by a union bound over all optima we obtain that the probability q_{10} that the algorithm finds an optimum in less than t iterations is

$$\begin{aligned}q_{10}&\le(1-q_5)2^{n^{1-\varepsilon}}\le 2\exp\left(-\frac{n^{1-c\ln 2-\alpha}}{8}\right)\exp\left(n^{1-\varepsilon}\ln 2\right)\\&=2\exp\left(n^{1-\varepsilon}\ln 2-\frac{n^{1-c\ln 2-\alpha}}{8}\right).\end{aligned}$$

This probability q_{10} tends to zero with growing n if and only if the argument of the exponential function tends to negative infinity. It does so if and only if α and c satisfy $\alpha+c\ln 2<\varepsilon$. Since ε is a positive constant, we can choose $\alpha:=\varepsilon/2$ and $c:=\varepsilon/2$ to satisfy this condition. □

The actual reason that the algorithm cannot find an optimum faster than in $\Omega(\frac{n\log n}{\lambda})$ iterations is the coupon collector effect when the algorithm tries to flip the few wrong bits left in the end of the optimization. However, if we have $2^{\Theta(n)}$ optima, the algorithm avoids this effect. To illustrate this idea consider the $(1+1)$ EA that optimizes the ONEMAX function, but the bit-strings with less than cn zero-bits, where c is some small constant, are considered optimal. Thus, this functions has no more than $O(2^{c\log_2(1/c)n})\subseteq 2^{\Theta(n)}$ optima. Clearly, the runtime of the $(1+1)$ EA on such function is linear, which may be proven with simple additive drift argument.

The following two theorems extend our $\Omega(\frac{n\mu}{\lambda})$ and $\Omega(\frac{n\log\log\frac{\lambda}{\mu}}{\log\frac{\lambda}{\mu}})$ bounds to the functions with $2^{O(n)}$ optima.

Theorem 12. *If μ is at most polynomial in n, then the $(\mu+\lambda)$ EA optimizes any pseudo-Boolean function with at most $2^{\beta n}$ optima, where β is some constant less than $\frac{1}{16\ln 2}$, in $\Omega(\frac{\mu n}{\lambda})$ iterations. If $\frac{\lambda}{\mu} > e^e$, then the stronger bound $\Omega(\frac{n\log\log\frac{\lambda}{\mu}}{\log\frac{\lambda}{\mu}})$ holds.*

Proof. By (20) the probability that the algorithm finds a particular optimum in less than $t := \lfloor\frac{\mu n}{8e\lambda}\rfloor$ iterations is

$$q_8 \leq \frac{\mu}{e^{\frac{n}{16}}-1} + \frac{\mu^2 n}{8e\lambda}\left(\frac{1}{2}\right)^{\frac{n}{4}}.$$

By a union bound taken over no more than $2^{\beta n}$ optima, the probability q_{11} that the algorithm finds any optimum in this time is

$$q_{11} \leq q_8 2^{\beta n} \leq \frac{\mu e^{(\ln 2)\beta n - \frac{n}{16}}}{1-e^{-\frac{n}{16}}} + \frac{\mu^2 n}{8e\lambda} 2^{\beta n - \frac{n}{4}}.$$

Since $\beta < \frac{1}{16\ln 2}$ and β is a constant, we have both $(\ln 2)\beta n - \frac{n}{16} < 0$ and $\beta n - \frac{n}{4} < 0$ (and both of them are linear in n). Thus, q_{11} tends to zero with growing n. Hence, the expected runtime of the algorithm is $\Omega(t) = \Omega(\frac{\mu n}{\lambda})$.

To prove the $\Omega(\frac{n\log\log\frac{\lambda}{\mu}}{\log\frac{\lambda}{\mu}})$ bound we argue in a similar way. By (22) the probability that the algorithm finds a particular optimum in less than $t := \lfloor\frac{(e-2)n\ln\ln\frac{\lambda}{\mu}}{4(e+1)\ln\frac{\lambda}{\mu}}\rfloor$ iterations is

$$q_9 \leq \frac{\mu}{e^{\frac{n}{16}}-1} + \mu\frac{(e-2)n\ln\ln\frac{\lambda}{\mu}}{4(e+1)\ln\frac{\lambda}{\mu}}\left(\ln\frac{\lambda}{\mu}\right)^{-n/4e}.$$

By a union bound taken over no more than $2^{\beta n}$ optima, the probability q_{12} that the algorithm finds any optimum in this time is

$$q_{12} \leq q_9 2^{\beta n} \leq \frac{\mu e^{\beta n\ln 2 - n/16}}{1-e^{-\frac{n}{16}}} + \mu\frac{(e-2)n\ln\ln\frac{\lambda}{\mu}}{4(e+1)\ln\frac{\lambda}{\mu}} e^{\beta n\ln 2 - n/4}.$$

Since $\beta < \frac{1}{16\ln 2}$ and β is a constant, we have both $(\ln 2)\beta n - \frac{n}{16} < 0$ and $\beta n\ln 2 - \frac{n}{4} < 0$ (and both of them are linear in n). Thus, q_{12} tends to zero with growing n. Hence, the expected runtime of the algorithm is $\Omega(t) = \Omega(\frac{n\log\log\frac{\lambda}{\mu}}{\log\frac{\lambda}{\mu}})$. □

2.2.4 Analysis of the $(\lambda \overset{1:1}{+} \lambda)$ EA

In this section we prove that our results (both upper bound from Theorem 8 and lower bound from Theorem 10) hold in an analogous fashion also for the $(\lambda \overset{1:1}{+} \lambda)$ EA, that is, we show

that this algorithm optimizes ONEMAX in an expected number of $\Theta(\frac{n\log n}{\lambda} + n)$ iterations. This improves over the $O(\frac{n\log n}{\lambda}+n\log\lambda)$ proven bound and the $O(\frac{n\log n}{\lambda}+n\log\log n)$ conjecture of [13].

Due to the differences in the algorithms, to prove our results we obviously cannot just apply the previous theorems in this work to the case $\lambda = \mu$. We recall that the $(\lambda \overset{1:1}{+} \lambda)$ EA uses a different parent selection. While the classic $(\mu + \lambda)$ EA chooses each parent independently and uniformly at random from the μ individuals, the $(\lambda \overset{1:1}{+} \lambda)$ EA creates exactly one offspring from each parent (see Algorithm 2). We note that the authors of [13] also use a slightly different way of selecting the next parent population. In principle, they take as new parent population the μ best individuals among parents and offspring (plus-selection). If this would lead to a new parent population only consisting of offspring, they remove the weakest offspring and replace it with the strongest individual from the previous parent population. Since this appears to be a not very common way of selecting the new population, we shall work with the classic plus-selection, favoring offspring in case of ties, and breaking further ties randomly (though, indeed, the tie-breaking is not important when optimizing ONEMAX via unary unbiased black-box algorithms). We note without proof that the following results and proofs are valid for the precise algorithm regarded in [13] as well.

We start by proving the upper bound for the runtime.

Theorem 13. *The expected runtime of the* $(\lambda \overset{1:1}{+} \lambda)$ *EA on the* ONEMAX *function is* $O(\frac{n\log n}{\lambda} + n)$.

Proof. We aim at adapting Theorem 8 for the $(\lambda \overset{1:1}{+} \lambda)$ EA. For this purpose we note that the proof of Theorem 8 only depends on the expected level improvement times $E[\tilde{T}_i]$ computed in Corollary 1, which again depend on the times needed for increasing the number of fit individuals computed in Lemma 32. Therefore, it suffices to show that the estimates of Lemma 32 and Corollary 1 are also valid for the $(\lambda \overset{1:1}{+} \lambda)$ EA.

We prove that Lemma 32 holds for the $(\lambda \overset{1:1}{+} \lambda)$ EA by observing that the probability $p_2(j)$ to create at least one copy of the fit individual satisfies the same estimate as the one used for the $(\mu + \lambda)$ EA, which is (9), with $\mu = \lambda$. For the $(\lambda \overset{1:1}{+} \lambda)$ EA, $p_2(j)$ is at least the probability that at least one of the j fit parent individuals creates as offspring a copy of it. By Lemma 21 we have

$$p_2(j) \geq 1 - \left(1 - \left(1 - \frac{1}{n}\right)^n\right)^j \geq 1 - \left(1 - \frac{1}{2e}\right)^j \geq \frac{1}{1 + \frac{2e}{j}},$$

which is the same estimate as for the $(\mu + \lambda)$ EA (with $\mu = \lambda$).

To prove that Corollary 1 holds for the $(\lambda \overset{1:1}{+} \lambda)$ EA as well, it is sufficient to show that the probability $p''(i)$ to create a superior individual satisfies as well the estimate (11) in the case $\mu = \lambda$. The probability $p''(i)$ is at least the probability that for at least one of the $\mu_0(i)$ best individuals the offspring is better than its parent. Using Lemma 21 we calculate

$$\begin{aligned} p''(i) &\geq 1 - \left(1 - \frac{n-i}{n}\left(1 - \frac{1}{n}\right)^{n-1}\right)^{\mu_0(i)} \geq 1 - \left(1 - \frac{n-i}{en}\right)^{\mu_0(i)} \\ &\geq 1 - \frac{1}{1 + \frac{\mu_0(i)(n-i)}{ne}}, \end{aligned}$$

which is the same value as in Corollary 1 when $\mu = \lambda$. □

Comparing this bound with the bound $O(\frac{n\log n}{\lambda}+n\log\lambda)$ proven in [13] and the bound $O(\frac{n\log n}{\lambda}+n\log\log n)$ conjectured in the same work, we immediately see that ours is at least as strong as these two for all values of λ. For $\lambda=\omega(\frac{\log n}{\log\log n})$, our bound is asymptotically smaller than both the proven bound and the conjecture.

We now prove a matching lower bound, which agrees with the one of Theorem 10 in the case of $\mu=\lambda$.

Theorem 14. *If λ is polynomial in n then the expected runtime of the $(\lambda \overset{1:1}{+} \lambda)$ EA on the OneMax function is $\Omega(\frac{n\log n}{\lambda}+n)$.*

Proof. We show that the main arguments of the proof for this bound in Theorem 10 are also valid for this parent selection mechanism.

To prove the $\Omega(\frac{n\log n}{\lambda})$ bound we can repeat the arguments from Theorem 10 without any changes. One needs to prove this bound only for $\lambda < c\log n$ for some arbitrary small constant c. The main argument is that with high probability there is a set of bits which were in a wrong position in all initial individuals and that at least one of those bits was not flipped by any of $t\lambda$ applications of the mutation operator for some $t=\Theta(\frac{n\log n}{\lambda})$. This argument stays valid for the fair parent selection as well.

To prove the $\Omega(n)$ bound we consider the complete trees for the $(\lambda \overset{1:1}{+} \lambda)$ EA. Since in a run of the $(\lambda \overset{1:1}{+} \lambda)$ EA each individual in the population creates exactly one offspring, the complete trees now have a slightly different structure, namely each node of the tree has exactly one child at each time step (instead of λ children). In return, we cannot argue that each edge is present in the true family tree with probability at most $1/\mu$ only (so we assume that all these edges are in fact present). Since $\lambda=\mu$, these two effects cancel.

More precisely, following the proof of Theorem 10 we argue that with high probability $q_7 \geq \exp(-\frac{\lambda}{e^{\frac{n}{16}}-1})$ all initial individuals have at least $n/4$ wrong bits. Next, we argue that in an analogous fashion as in Theorem 6 – and this is where the two effects truly cancel – the probability q_{opt} that the optimum occurs in any tree in less than $t := \lceil \frac{n}{8e} \rceil$ iterations is at most

$$q_{opt} \leq \lambda \sum_{\ell=0}^{t} \binom{t}{\ell} p(\ell,n) \leq \lambda t \left(\frac{1}{2}\right)^{n/4}.$$

Since we only consider λ that is polynomial in n, this entity tends to zero, when n tends to infinity. Therefore, the probability that the algorithm finds an optimum in $t=\Theta(n)$ iterations is at most $(1-q_7)+q_7q_{opt}$ that is less than some constant, if n is large enough. Hence, the expected runtime of the $(\lambda \overset{1:1}{+} \lambda)$ EA is $\Omega(n)$. □

We note that we can extend this lower bound to functions with multiple optima in the same manner as in Subsection 2.2.3 for the $(\lambda \overset{1:1}{+} \lambda)$ EA. We omit the proof, since it repeats the proof of Theorems 11 and 12.

2.3 Analysis of the (μ, λ) EA

To gain a deeper understanding of the population dynamics of EAs in the case where there is no clear drift, we regard the (μ, λ) EA settings with λ close to $e\mu$. We prove three results inside this phase transition region $\lambda = (1 \pm \varepsilon)e\mu$.

We first show that the super-polynomial range already starts when $\lambda \leq (1 - \varepsilon)e\mu$ for $\varepsilon = \omega(n^{-1/2})$. To prove this result, we do not extend the general but technical negative-drift-in-populations theorem of [96] to smaller negative drifts, but instead use a basic drift argument. This approach avoids the use of family trees and branching processes and might thus be a light-weight alternative for similar analysis problems as well.

When μ is not overly large, namely $\mu \leq n^{1/2-c}$ for an arbitrary small constant $c > 0$, then the weaker condition $\lambda \leq e\mu$ suffices to lead to a super-polynomial runtime. Note that in this regime, we have essentially no drift in the sub-population of best individuals. The reason why the (μ, λ) EA still has difficulties to find the optimum is that in this no-drift regime, the number of best individuals performs an unbiased a random walk (with typical step sizes up to $\sqrt{\mu}$). When this walk reaches zero, no individual on this fitness level is left and the (μ, λ) EA, due to the limited population size, takes a non-trivial amount of time to re-generate such an individual. The time this walk takes to reach zero is roughly $O(\mu)$. Hence if $\mu \leq n^{1/2-c}$ and the best fitness in the population is close to n, then the $O(\mu)$ iterations with $O(\lambda) = O(\mu)$ offspring generated are not enough to produce a strictly better individual. For this reason, we exhibit here a (slow, namely constant per $O(\mu)$ iterations) negative drift in the fitness of the best individual in the population. This negative drift translates into a long runtime via a negative drift theorem.

When μ is slightly larger, namely at least $n^{2/3+c}$ for an arbitrary constant $c > 0$, then for all $\lambda \geq e\mu$, that is, again including settings with essentially no drift, we have a polynomial runtime of $O(n\lambda \log n)$, which means $O(n \log n)$ iterations. This is, the same runtime guarantee as shown for the constant $(1 + \varepsilon)$ drift case in [18] (which has been recently improved to $O(n)$ iterations in [17]), but the reasons are different. Here, we have essentially a no-drift regime. Hence the number of individuals on the highest fitness level performs an unbiased random walk. Different from above, the larger population sizes implies that before this walk reaches zero, some individuals are generated on a higher level. This is not the immediate pathway to the optimum since these small sub-populations have a good chance of dying out quickly (they perform the same type of unbiased random walk, but starting close to zero). The reason why these climbers make a difference is that they stabilize the fitness level below them. We recall that such an individual, when chosen as parent, creates an equally fit offspring with probability roughly $\frac{1}{e}$. In addition, with probability $\frac{1}{e}$ it creates an offspring on the next lower fitness level. These offspring create a positive drift in this level and hinder it from dying out after $O(\mu)$ iterations. Consequently, this lower level has ample time to create further climbers until one of them successfully take over the population.

The results of this section are summarized in Figure 7 together with previous results already shown in Figure 4.

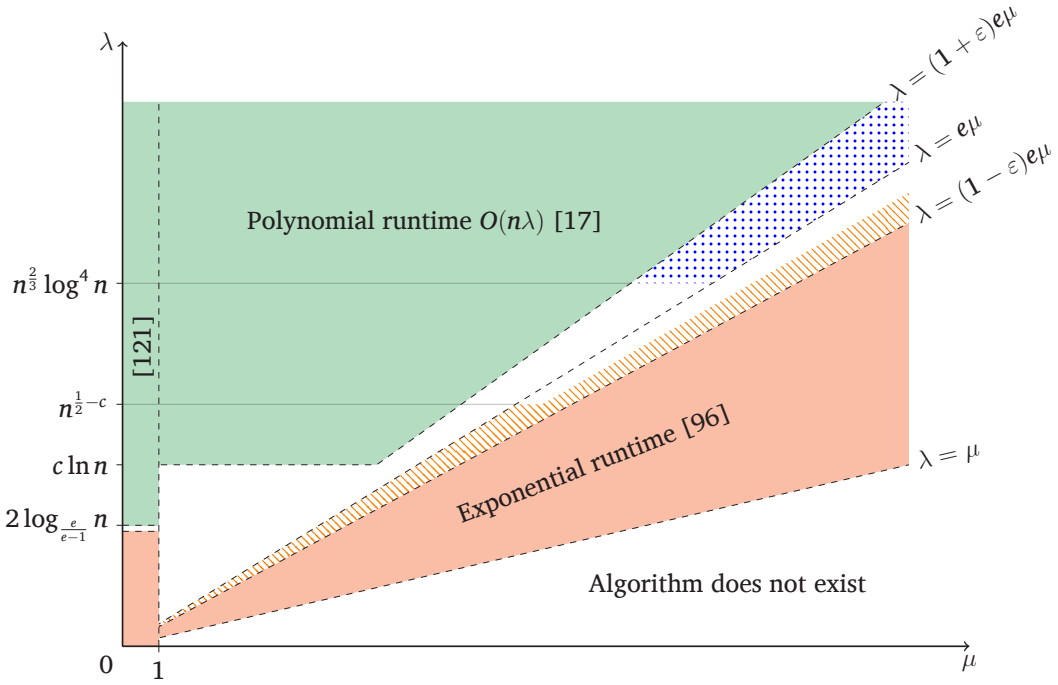


Figure 7 – *Our results for the (μ, λ) EA in its parameters space. We proved the super-polynomial runtime in a zone marked with orange lines and we proved the polynomial runtime in a zone marked with blue dots. Although our results cover only a small range of parameters, we revealed the most interesting regimes of the (μ, λ) EA.*

2.3.1 Lower Bounds for $\lambda \leq (1-\varepsilon)\mu e$ with $\varepsilon = \omega\left(\frac{1}{\sqrt{n}}\right)$.

In this section, we show that when $\lambda \leq (1-\varepsilon)\mu e$ for some $\varepsilon = \omega\left(\frac{1}{\sqrt{n}}\right)$, then the runtime of the (μ, λ) EA on OneMax is super-polynomial. Our analysis reproves the exponential runtime shown in [96] for constant ε and it enlarges the range for which a super-polynomial runtime is proven to $\varepsilon = \omega\left(\frac{1}{\sqrt{n}}\right)$.

Theorem 15. *Consider the run of the (μ, λ) EA on n-dimensional OneMax function.*

1) *If there exists a constant $\varepsilon \in (0, 1)$ such that $\lambda \leq (1-\varepsilon)\mu e$, then the expected runtime is exponential in n.*
2) *If there exists $\varepsilon = \omega(\frac{1}{\sqrt{n}})$ such that $\lambda \leq (1-\varepsilon)\mu e$, then the expected runtime is super-polynomial in n.*

To prove this result we use the lower bound version of the additive drift theorem. For this we define the potential function $g(x)$ both for individuals and for populations. The potential of a population is a sum of potentials of its individuals. The potential of an individual, roughly speaking, is exponential in its fitness. This lets us estimate the potential of the next population via the potential of all offspring, including those who do not survive. By this, we circumvent the usually difficult analysis of the effects of selection.

Exploiting that fitness gains are rare when close to the optimum, we show that this potential has an expected increase (*drift*) of at most 2λ per iteration. Again exploiting the strong growth of this potential function, we see that the potential difference of the initial population and any population containing the optimum is large, which gives the desired lower bound via the additive drift theorem.

The potential function g is defined as follows. Let $\tau := \frac{4e}{\varepsilon}$,

$$\alpha := 1 - \frac{1}{\tau} \ln\left(1 + \frac{1}{\tau}\right)$$

and $f_0 := \lceil \alpha n \rceil$. Note that α is at most 1 and it is at least

$$\alpha \geq 1 - \frac{1}{\tau^2} = 1 - \frac{\varepsilon^2}{16e^2} \geq 1 - \frac{1}{16e^2},$$

since $\ln(1+x) \leq x$ holds for all $x > -1$ and since $\varepsilon \leq 1$ (otherwise we have $\lambda < \mu$). The potential function g is defined over $\{0,1\}^n$ by

$$g(x) = \begin{cases} \tau^{f(x)-f_0} & \text{if } f(x) \geq f_0, \\ 0 & \text{otherwise.} \end{cases}$$

For a population P, we define

$$g(P) = \sum_{x \in P} g(x).$$

The following key lemma estimates the drift of the potential in each generation.

Lemma 40. *For all t, we have $E[g(P_{t+1})] \leq g(P_t) + 2\lambda$.*

To prove this result, we first compute the expected fitness of an offspring of a search point of fitness at least f_0.

Lemma 41. *For any individual x of fitness $f(x) \geq f_0$, we have*

$$E[g(\mathcal{M}x)] \leq \frac{1}{e}(1+\varepsilon)g(x),$$

where $\mathcal{M}$ stand for the operator of the standard bit mutation with mutation rate $\frac{1}{n}$.

Proof. Let x be an individual such that $f(x) \geq f_0$ and let $d := n - f(x)$ be its distance to the optimum. Note that

$$d \leq (1-\alpha)n = \frac{n}{\tau} \ln\left(1 + \frac{1}{\tau}\right) \leq \frac{n}{\tau^2}, \tag{24}$$

where we used the inequality $x \geq \ln(1+x)$ which holds for all $x > -1$. Let $\delta_x = f(\mathcal{M}x) - f(x)$ We aim at showing that

$$g(\mathcal{M}x) = g(\mathcal{M}x)\mathbb{1}_{(\delta_x > 0)} + g(\mathcal{M}x)\mathbb{1}_{(\delta_x < 0)} + g(\mathcal{M}x)\mathbb{1}_{(\delta_x = 0)}$$

and analyze each term separately.

Positive δ_x. By the definition of g we compute

$$E[g(\mathcal{M}x)\mathbb{1}_{(\delta_x>0)}] = g(x)\sum_{i=1}^{d}\Pr[\delta_x = i]\tau^i.$$

By Lemma 10,

$$\begin{aligned}\sum_{i=1}^{d}\Pr[\delta_x = y]\tau^i &\le \sum_{i=1}^{d}\binom{d}{y}\left(\frac{1}{n}\right)^i\tau^i \\ &= \left(1+\frac{\tau}{n}\right)^d - 1 \le \exp\left(\frac{d\tau}{n}\right) - 1.\end{aligned}$$

Since by (24) we have $d \le \frac{n}{\tau}\ln\left(1+\frac{1}{\tau}\right)$, we also have

$$\begin{aligned}E[g(\mathcal{M}x)\mathbb{1}_{(\delta_x>0)}] &\le \left(\exp\left(\frac{d\tau}{n}\right) - 1\right)g(x) \\ &\le \left(\exp\left(\ln\left(1+\frac{1}{\tau}\right)\right) - 1\right)g(x) = \frac{g(x)}{\tau}.\end{aligned}$$

Negative δ_x. We compute

$$\begin{aligned}E[g(\mathcal{M}x)\mathbb{1}_{(\delta_x<0)}] &= g(x)\sum_{i=f_0-f(x)}^{-1}\Pr[\delta_x = i]\tau^i \\ &\le \frac{g(x)}{\tau}\sum_{i<0}\Pr[\delta_x = i] \le \frac{g(x)}{\tau}.\end{aligned}$$

Zero δ_x. By Lemma 11 we have

$$E[g(\mathcal{M}x)\mathbb{1}_{(\delta_x=0)}] = g(x)\Pr[f(\mathcal{M}x) = f(x)] \le g(x)\frac{1}{e\left(1-\frac{d(n-d)}{(n-1)^2}\right)}.$$

Since $d \le \frac{n}{\tau^2}$, we have

$$E[g(\mathcal{M}x)\mathbb{1}_{(\delta_x=0)}] \le g(x)\frac{1}{e\left(1-\frac{n}{(n-1)\tau^2}\right)}.$$

Note that we also have

$$\begin{aligned}\frac{1}{1-\frac{n}{(n-1)\tau^2}} &= \frac{1+\frac{1}{\tau^2}}{\left(1-\frac{n}{(n-1)\tau^2}\right)\left(1+\frac{1}{\tau^2}\right)} = \frac{1+\frac{1}{\tau^2}}{1+\frac{1}{\tau^2}-\frac{n}{(n-1)\tau^2}-\frac{n}{(n-1)\tau^4}} \\ &\le 1+\frac{1}{\tau^2}.\end{aligned}$$

Therefore,

$$E[g(\mathcal{M}x)\mathbb{1}_{(\delta_x=0)}] \le g(x)\frac{1}{e}\left(1+\frac{1}{\tau^2}\right)$$

Finally, we obtain

$$\begin{aligned}E[g(\mathcal{M}x)] &= E\left[g(\mathcal{M}x)\mathbb{1}_{(\delta_x>0)}\right] + E\left[g(\mathcal{M}x)\mathbb{1}_{(\delta_x<0)}\right] \\ &\quad + E\left[g(\mathcal{M}x)\mathbb{1}_{(\delta_x=0)}\right] \\ &\leq g(x)\left(\frac{1}{e} + \frac{2}{\tau} + \frac{1}{e\tau^2}\right) = g(x)\frac{1}{e}\left(1 + \frac{\varepsilon}{2e} + \frac{\varepsilon^2}{16e^3}\right) \\ &\leq g(x)\frac{1}{e}(1+\varepsilon).\end{aligned}$$

□

Since Lemma 41 applies when $f(x) \geq f_0$, we now show that the expected potential of an offspring of a search point of fitness at most $f_0 - 1$ is at most constant.

Lemma 42. *If n is large enough, for all individuals x such that $f(x) < f_0$, we have*

$$E[g(\mathcal{M}x)] \leq 2.$$

Proof. Let x be an individual such that $f(x) < f_0$. Then $\mathcal{M}x$ has potential zero unless it gains at least the difference between $f(x)$ and f_0, which is at least 1. By the law of total probability and by Lemma 10, we have

$$\begin{aligned}E[g(\mathcal{M}x)] &= \sum_{i=f_0-f(x)}^{n-f(x)} \Pr[\delta_x = k]\tau^{(i-f_0+f(x))} \\ &\leq \tau^{f(x)-f_0} \sum_{i=f_0-f(x)}^{n-f(x)} \binom{n-f(x)}{k}\left(\frac{1}{n}\right)^i \tau^i \\ &\leq \tau^{f(x)-f_0}\left(1+\frac{\tau}{n}\right)^{n-f(x)}.\end{aligned}$$

This expression monotonically grows with the growth of $f(x)$ (since increasing $f(x)$ by a small δ increases it by a factor of $(\tau(1+\frac{\tau}{n})^{-1})^\delta > 1$), hence it is at most

$$\begin{aligned}E[g(\mathcal{M}x)] &\leq \frac{1}{\tau}\left(1+\frac{\tau}{n}\right)^{n-f_0-1} \leq \left(1+\frac{\tau}{n}\right)^{(1-\alpha)n} = \left(1+\frac{\tau}{n}\right)^{\frac{n}{\tau}\ln(1+\frac{1}{\tau})} \\ &\leq e^{\ln(1+\frac{1}{\tau})} = 1 + \frac{1}{\tau} \leq 2.\end{aligned}$$

□

Now we prove Lemma 40 using Lemmas 41 and 42.

Proof of Lemma 40. Let y be an offspring created from P_t. Then we have

$$\begin{aligned}E[g(y)] &= \sum_{x\in P_t} \frac{E[g(\mathcal{M}x)]}{\mu} \leq \sum_{x\in P_t} \frac{\frac{1}{e}(1+\varepsilon)g(x) + 2}{\mu} \\ &= \frac{\frac{1}{e}(1+\varepsilon)g(P_t)}{\mu} + 2.\end{aligned}$$

Let $\tilde{P}_{t+1}$ be the set of the λ offspring generated from P_t. Since $P_{t+1} \subset \tilde{P}_{t+1}$, we have

$$E[g(P_{t+1})] \leq E[g(\tilde{P}_{t+1})] \leq \frac{\lambda}{e\mu}(1+\varepsilon)g(P_t) + 2\lambda.$$

Recall that $\lambda \leq (1-\varepsilon)\mu e$, hence

$$\frac{\lambda}{e\mu}(1+\varepsilon)g(P_t) + 2\lambda \leq (1-\varepsilon^2)g(P_t) + 2\lambda \leq g(P_t) + 2\lambda.$$

□

Now we are almost in position to apply the additive drift theorem to potential function $g(P_t)$. For this we note that if the algorithm has found the optimum x^*, then $g(P_t) \geq g(x^*) = \tau^{n-f_0}$. Therefore, it is sufficient to show that the expected time for the potential to reach τ^{n-f_0} is exponential. To use the additive drift, however, we must ensure that we do not jump over the target. Hence we define the following random process

$$Z_t := \min\{\tau^{n-f_0}, g(P_t)\}.$$

We also define $T' := \inf\{t \geq 0 \mid Z_t = 0\}$ and $\mathcal{S}$ by the state space of the random process Z_t. Note that if T is the runtime of the algorithm, we have $E[T'] \leq E[T]$.

To apply the additive drift theorem to Z_t we first find the expected initial value Z_0 in the following lemma.

Lemma 43. *If n is large enough, and if μ is sub-exponential in n, we have*

$$E[Z_0] \leq \frac{1}{2}\tau^{n-f_0}.$$

Proof. We have

$$E[Z_0] \leq E[g(P_0)] = \mu E[g(x)],$$

where x is an individual chosen uniformly at random from $\{0,1\}^n$. The fitness of x follows the binomial distribution with parameters n and $\frac{1}{2}$ with $E[f(x)] = \frac{n}{2}$. Hence by Chernoff bounds the probability that $f(x) \geq f_0 = \alpha n$ is

$$\Pr[f(x) \geq \alpha n] = \Pr\left[f(x) \geq (1 + (2\alpha - 1))\frac{n}{2}\right] \leq \exp\left(-\frac{(2\alpha-1)n}{6}\right).$$

Since $\alpha \geq 1 - \frac{1}{16e^2} > \frac{7}{8}$, we have

$$\Pr[f(x) \geq \alpha n] \leq e^{-\frac{n}{8}}.$$

Since the maximal potential of an individual is τ^{n-f_0}, and all individuals with fitness less than αn have zero potential, we have

$$\mu E[g(x)] \leq \mu e^{-\frac{n}{8}}\tau^{n-f_0} \leq \frac{1}{2}\tau^{n-f_0},$$

if n is large enough, since μ is sub-exponential in n. □

Now we are in position to prove the main result of this subsection.

Proof of Theorem 15. If μ is super-polynomial, the expected runtime is also super-polynomial, hence we assume that μ and λ are at most polynomial.

We transform Lemma 40 for Z_t. For all $t \geq 0$ and for all $s \in \mathcal{S} \backslash \{\tau^{n-f_0}\}$, we have

$$E[Z_{t+1} - Z_t \mid Z_t = s] \leq E[g(P_{t+1}) - g(P_t) \mid Z_t = s] \leq 2\lambda.$$

By the additive drift theorem and Lemma 43, we have

$$\begin{aligned} E[T] \geq E[T'] &\geq \frac{\tau^{n-f_0} - E[Z_0]}{2\lambda} \geq \frac{\tau^{n-f_0}}{4\lambda} \geq \frac{1}{4\tau\lambda} \exp\left(\frac{n}{\tau} \ln\left(1 + \frac{1}{\tau}\right) \ln \tau\right) \\ &\geq \frac{1}{4\tau\lambda} \exp\left(\frac{n}{\tau}\left(\frac{1}{\tau} - \frac{1}{2\tau^2}\right) \ln \tau\right) \\ &\geq \frac{1}{4\tau\lambda} \exp\left(\frac{n}{2\tau^2} \ln \tau\right) = \frac{\varepsilon}{16e\lambda} \exp\left(\frac{n\varepsilon^2}{32e^2} \ln \frac{4e}{\varepsilon}\right). \end{aligned}$$

If ε is a constant, then the expected runtime is exponential. If $\varepsilon = \omega\left(\frac{1}{\sqrt{n}}\right)$, we have two cases.

Case 1. When $\varepsilon \geq n^{-\frac{1}{4}}$, we have

$$E[T] \geq \frac{1}{16e\lambda n^{\frac{1}{4}}} \exp\left(\frac{\sqrt{n}}{32e^2}\right).$$

Case 2. For $\varepsilon \leq n^{-\frac{1}{4}}$, we have

$$E[T] \geq \frac{1}{16e\lambda\sqrt{n}} \exp\left(\omega(1) \ln n\right).$$

In both cases the expected runtime is super-polynomial. □

2.3.2 The Runtime when $\lambda \leq \mu e$.

The Irrationality of e and its Consequences on the Runtime.

In this subsection, we show that due to the irrationality of the Euler's number if the population size μ is too small, we cannot approximate e with $\frac{\lambda}{\mu}$ close enough, hence $\lambda \leq e\mu$ also implies that $\lambda \leq (1 - \omega(\frac{1}{\sqrt{n}}))e\mu$ and therefore, we can apply the results of the previous section.

Theorem 16. *Let $\lambda \leq e\mu$ and let μ be a function of n which tends to positive infinity when n tends to positive infinity. If there exists a constant $c \in (0, \frac{1}{4})$ such that $\mu \leq n^{\frac{1}{4}-c}$, then the expected runtime of the (μ, λ) EA on n-dimensional ONEMAX is super-polynomial in n.*

To prove this theorem we use the definition of the irrationality measure. Let $x \in \mathbb{R}$ and let R be a set of positive numbers d such that the set

$$\left\{p, q \in \mathbb{N} \text{ such that } \left|x - \frac{p}{q}\right| \leq \frac{1}{q^d}\right\}$$

is finite. The *irrationality measure* of x is the infimum of R.

Theorem 17 (Theorem 1 in [22]). *The irrationality measure of e is* 2.

From Theorem 17 and Theorem 15 we deduct the proof of Theorem 16.

Proof of Theorem 16. Let $d := 2+c$. By the definition of irrationality measure and by Theorem 17, the set $\{\lambda, \mu \in \mathbb{N} \mid \left|e - \frac{\lambda}{\mu}\right| \leq \frac{1}{\mu^d}\}$ is finite. Therefore, if μ is large enough, we have

$$\left|e - \frac{\lambda}{\mu}\right| \geq \frac{1}{\mu^d} \geq \frac{1}{n^{(\frac{1}{4}-c)(2+c)}} = n^{-\frac{1}{2}+\frac{7c}{4}+c^2} = \omega\left(\frac{1}{\sqrt{n}}\right).$$

Therefore,

$$\lambda = \left(1 - \frac{1}{e}\left(e - \frac{\lambda}{\mu}\right)\right) e\mu = \left(1 - \omega\left(\frac{1}{\sqrt{n}}\right)\right) e\mu.$$

Hence, by Theorem 15, the expected runtime of the (μ, λ) EA on n-dimensional OneMax is super-polynomial in n. □

The Super-polynomial Runtime for Low μ.

In this subsection we assume that the population size is bounded, and that $\frac{\lambda}{\mu}$ is as close to e as possible. More precisely,

$$\begin{cases} \lambda \leq e\mu, \\ \text{for all } \varepsilon = \omega\left(\frac{1}{\sqrt{n}}\right) \text{ we have } \lambda \geq (1-\varepsilon)e\mu \\ \text{there exists a constant } c \in (0, \frac{1}{2}) \text{ such that } \mu \leq n^{\frac{1}{2}-c}, \\ \mu \to \infty \text{ when } n \to \infty. \end{cases} \tag{25}$$

The main result of this subsection is the following theorem.

Theorem 18. *If the conditions* (25) *are satisfied, then the expected runtime of the* (μ, λ) *EA on n-dimensional OneMax is super-polynomial in n.*

Since the proof of Theorem 18 is quite technical, we first sketch it so that it was easier for the reader to follow our arguments. We use the following notation in the rest of this section. We define the *top level* $f_{\text{top}}(t)$ at the generation t as the best fitness among the population. Namely, $f_{\text{top}}(t) := \max_{x \in P_t} f(x)$. By X_t we denote the number of individuals of fitness f_{top} after the t-th generation.

We split an algorithm run into phases of variable duration, but not longer than some $L = \Theta(\mu)$ generations (we will define L precisely later), such that the increase of f_{top} can appear only in the last generation of such phase. We consider the fitness in the first generation of each phase and show that if f_{top} is large enough, this fitness has a constant probability to be smaller than the the fitness in the beginning of the previous phase, while the probability that it is greater is too small. Therefore, this fitness at the beginning of each phase is a subject of the negative drift theorem (Theorem 1), which gives us the desired result.

In order to show that a phase of length L results into decrease of the fitness, we first consider the probability of the event that in L consecutive generations no superior individual

is created (Lemma 44). Then, conditional on this event we consider the number X_t of the best offspring in the population. We show that its conditional distribution is almost the same as unconditional in Lemma 45. Hence, X_t has almost not drift (or the drift is small compared to its random deviations). We apply our methods from Subsection 2.1.2 to X_t to show that it reaches zero in at most L iterations with at least constant probability in Lemma 46.

We start with estimating the probability of the event that no good mutation occurs during L generations in a row. For any $L \in \mathbb{N}$ we define N_L as an event when during L consecutive generations, the following two conditions are satisfied.

1) For all individuals x of fitness $f(x) \leq f_{\text{top}} - 1$ we have $f(\mathcal{M}x) < f_{\text{top}}$,
2) for all individuals x of fitness $f(x) = f_{\text{top}}$, we have $f(\mathcal{M}x) \leq f_{\text{top}}$ with $f(\mathcal{M}x) = f_{\text{top}}$ if and only if $\mathcal{M}x$ is the exact copy of x.

We call an individual x *breaking*, if it is either an offspring of individual y with $f(y) < f_{\text{top}}$ and $f(x) \geq f_{\text{top}}$ or if it is an offspring of an individual y with $f(y) = f_{\text{top}}$ and x is not an exact copy of y. In other words, this is an individual which breaks the conditions of event N_L. We show the following lower bound on the probability of N_L.

Lemma 44. *Assume that the conditions* (25) *are satisfied. Let* $D := n^c$ *and* $L \in \mathbb{N}$. *Then if* $f_{\text{top}} \geq n - D + 1$ *and if* n *is large enough, we have*

$$\Pr(N_L) \geq 1 - \frac{eL}{\sqrt{n}}.$$

This bound is independent of the individuals which are chosen as parents during these L generations.

Proof. We first compute the probability that we create a breaking individual. In each individual of P_t with fitness f_{top} there are $n - f_{\text{top}} \leq D - 1$ bits in wrong positions. If such an individual is selected as a parent, the probability that the standard bit mutation (which flips each bit independently with probability $\frac{1}{n}$) does not flip any of these bits is at least $(1 - \frac{1}{n})^{D-1}$.

In each individual of P_t with fitness less than f_{top} there are at least $n - f_{\text{top}} - 1$ bits in wrong position. Let us fix $n - f_{\text{top}} - 1 \leq D$ of these wrong bit positions. If we select an individual x with $f(x) < f_{\text{top}}$ as a parent, and then the standard bit mutation does not flip any of the bits in these positions, then the fitness of the offspring is not greater than $f_{\text{top}} - 1$. The probability that none of these bits are flipped is at least $(1 - \frac{1}{n})^D$.

Hence, independently of the choice of the parent, the probability to crate a non-breaking offspring is at least $(1 - \frac{1}{n})^D$. In each generation we create λ offspring independently, hence the probability that we do not create any breaking offspring in L generations is

$$\Pr(N_L) \geq \left(1 - \frac{1}{n}\right)^{D\lambda L}.$$

By Bernoulli's inequality we have

$$\Pr(N_L) \geq 1 - \frac{D\lambda L}{n} \geq 1 - \frac{n^c e n^{\frac{1}{2}-c} L}{n} \geq 1 - \frac{eL}{\sqrt{n}}.$$

□

Now when we see that event N_L is quite likely to happen we consider how the number of the best individuals in the population X_t changes through L generations. Normally, $X_{t+1} \sim \min\{\text{Bin}(\lambda, \frac{X_t}{e\mu}), \mu\}$, but conditioning on N_L can change its distribution. We show that this change is almost negligible in the following lemma.

Lemma 45. *Assume that the conditions* (25) *are satisfied. Let* $D := n^c$ *and* $f_{\text{top}} \geq n - D + 1$. *Conditional on event* N_1, *there exists a sequence* $(p_n)_{\mathbb{N}}$ *such that for all* n, *we have* $X_{t+1} \sim \min\{\text{Bin}(\lambda, p_n), \mu\}$ *and* $\left| p_n - \frac{X_t}{e\mu} \right| \leq \frac{2X_t}{e\mu\sqrt{n}}$.

Proof. Let $\tilde{X}_{t+1}$ be the number of offspring with fitness f_{top}. If we condition on N_1, all $\tilde{X}_{t+1}$ such offspring are the copies of their parents. Therefore, $\tilde{X}_{t+1}$ follows a binomial distribution $\text{Bin}(\lambda, p_n)$, where p_n is the probability to choose one of X_t individuals as a parent and then create its copy conditional on N_1.

Consider a creation of an offspring. Let A be the event when an individual of fitness f_{top} is picked as a parent and let B be the event when we create an exact copy of the chosen parent. Let C the event that the created offspring is non-breaking. Note that events A and B are independent, hence $\Pr[A \cap B] = \Pr[A] \Pr[B]$. Also, since all λ offspring are created independently we have $\Pr[A \cap B \mid N_1] = \Pr[A \cap B \mid C]$. By the Bayes' theorem we have

$$\begin{aligned} p_n &= \Pr[A \cap B \mid N_1] = \Pr[A \cap B \mid C] \\ &= \Pr[A \cap B] \frac{\Pr[C \mid A \cap B]}{\Pr[C]} = \Pr[A] \Pr[B] \frac{1}{\Pr[C]}, \end{aligned}$$

since event $A \cap B$ is a sub-event of C. By the same arguments as in Lemma 44 and by Lemma 19 we obtain

$$\begin{aligned} \Pr[C] &\geq \left(1 - \frac{1}{n}\right)^D \geq \left(1 - \frac{D}{n} - o\left(\frac{D}{n}\right)\right) \\ &= (1 - n^{c-1} - o(n^{c-1})) \geq 1 - \frac{1}{\sqrt{n}}, \end{aligned}$$

if n is large enough.

We have $\Pr[A] = \frac{X_t}{\mu}$ and $\Pr[B] = (1 - \frac{1}{n})^n$. Therefore, by Lemma 19 we obtain

$$p_n \geq \Pr[A] \Pr[B] = \left(1 - \frac{1}{n}\right)^n \frac{X_t}{\mu} \geq \frac{X_t}{e\mu} \left(1 - O\left(\frac{1}{n}\right)\right), \tag{26}$$

and

$$\begin{aligned} p_n &\leq \frac{\Pr[A] \Pr[B]}{\Pr[C]} \leq \left(1 - \frac{1}{n}\right)^n \frac{X_t}{\mu} \left(1 - \frac{1}{\sqrt{n}}\right)^{-1} \\ &\leq \frac{X_t}{e\mu} \left(1 + \frac{1}{\sqrt{n}} + o\left(\frac{1}{\sqrt{n}}\right)\right) \leq \frac{X_t}{e\mu} \left(1 + \frac{2}{\sqrt{n}}\right), \end{aligned} \tag{27}$$

if n is large enough, where we used the Taylor's formula for function $f(x) = \frac{1}{1-x}$. Hence, by (26) and (27) we conclude

$$\left| p_n - \frac{X_t}{e\mu} \right| \leq \frac{X_t}{e\mu} \max\left\{ O\left(\frac{1}{n}\right), \frac{2}{\sqrt{n}} \right\} = \frac{2X_t}{e\mu\sqrt{n}},$$

if n is large enough. □

Lemma 45 shows that X_t conditional on N_1 satisfies the conditions of Theorem 7 (the drift analysis of processes with no drift). Hence, the probability that X_t reaches zero in $L := 40\mu + 1$ iterations (conditional on N_L) is at least some constant s. We make this probability unconditional in the following lemma.

Lemma 46. *Let $L = 80\mu + 1$. Assume that conditions* (25) *are satisfied and let $D = n^c$. Then, if $f_{\text{top}} \geq n - D + 1$ and if n is large enough, the probability to lose the top level after L generations is at least $\frac{3s}{4}$, where s is a constant from Theorem 7.*

Proof. By Lemma 44 we have

$$\Pr(N_L) \geq 1 - \frac{eL}{\sqrt{n}} = 1 - \frac{80e\mu + e}{\sqrt{n}} \geq 1 - \frac{80e}{n^c} - \frac{e}{\sqrt{n}} = 1 - o(1).$$

Hence, if n is large enough, $\Pr(N_L) \geq \frac{3}{4}$. By Lemma 45 and Theorem 7 the probability to lose all individuals in L iterations conditional on N_L is at least s. Therefore, denoting by A the event that we lose all best individuals after L generations, we have

$$\Pr[A] \geq \Pr[A \mid N_L] \Pr[N_L] \geq \frac{3s}{4}.$$

□

Now we are going to split the algorithm run into phases of variable length. We define $\phi(0) = 0$ and, for all $t \in \mathbb{N}$,

$$\phi(t+1) = \min\left\{\phi(t) + L, \min\{\tau \geq \phi(t) \mid f_{\text{top}}(\tau + L) > f_{\text{top}}(\phi(t))\}\right\},$$

where we recall that in our notation a minimum of an empty set is the positive infinity, and $Z_t := f_{\text{top}}(\phi(t))$. In other words, we divide the process into phases of variable lengths so that if the top level does not increase during the L next generations, the phase length is L generations. Otherwise, the phase is stopped as soon as the top level exceeds its value. This way, during phase t (between Z_t and Z_{t+1}), there can only be one generation after which the top level goes from some $f_1 \leq f_{\text{top}}(\phi(t))$ to some $f_2 > f_{\text{top}}(\phi(t))$.

In order to apply the negative drift theorem to Z_t, let $b(n) = n$ and $a(n) = n - n^c$. In the following lemma we bound the probabilities of steps towards optimum.

Lemma 47. *Assume that the conditions* (25) *are satisfied. Then for all $i \geq 1$ we have*

$$\Pr[Z_{t+1} - Z_t = i \mid Z_t \geq a(n)] \leq L\lambda \frac{n^{-i(1-c)}}{i!}.$$

Proof. The probability $\Pr[Z_{t+1} - Z_t = i \mid Z_t > a(n)]$ is the probability that we crate at least one individual with fitness $f_{\text{top}}(\phi(t)) + i$ in L generations. Consider the probability to create such offspring. Assume that we choose an individual with fitness $f(x) \leq f_{\text{top}}(\phi(t))$ as a parent. Let $d := n - f_{\text{top}}(\phi_t)$ and let $\delta := f_{\text{top}}(\phi_t) - f(x)$. By Lemma 10, the probability to create such offspring is

$$\begin{aligned}\Pr[f(\mathcal{M}x) = f_{\text{top}} + i] &\leq \binom{d+\delta}{i+\delta}\frac{1}{n^{i+\delta}} \leq \frac{d^i}{i!n^i} \cdot \frac{(d+\delta)^\delta}{i^\delta n^\delta} \\ &\leq \frac{d^i}{i!n^i} \leq \frac{n^{ic}}{i!n^i} = \frac{n^{-i(1-c)}}{i!}.\end{aligned}$$

By the union bound over all λL offspring which we create in λL iterations we have

$$\Pr[Z_{t+1} - Z_t = i \mid Z_t \geq a(n)] \leq L\lambda \Pr[f(\mathcal{M}x) = f_{\text{top}} + i] \leq L\lambda \frac{n^{-i(1-c)}}{i!}.$$

□

Now we are in position to prove Theorem 18.

Proof of Theorem 18. To apply the negative drift theorem (Theorem 1), we define $\gamma := c \ln n - \ln 81 + \ln \frac{s}{4} - 1$, where s is the constant from Theorem 7. We compute the expectation of the exponential drift as follows (implicitly we compute it conditional on $Z \in (a(n), b(n))$, but we omit this notation to make it easier to read).

$$E\left[e^{\gamma(Z_{t+1}-Z_t)}\right] \leq e^{-\gamma} \Pr[Z_{t+1} < Z_t] + e^0 \Pr[Z_{t+1} = Z_t] + \sum_{i \geq 1} e^{\gamma i} \Pr[Z_{t+1} - Z_t = i].$$

We compute each term separately. First, by Lemma 46 we have

$$e^{-\gamma} \Pr[Z_{t+1} < Z_t] \leq 81 \frac{s}{4} n^{-c} \cdot \frac{3s}{4} = O(n^{-c}) = o(1).$$

We also have

$$e^0 \Pr[Z_{t+1} = Z_t] \leq 1 - \Pr[Z_{t+1} < Zt] \leq 1 - \frac{3s}{4}.$$

For the third term by Lemma 47 we compute

$$\begin{aligned}
\sum_{i \geq 1} e^{\gamma i} \Pr[Z_{t+1} - Z_t = i] &\leq \sum_{i \geq 1} e^{\gamma i} L\lambda \frac{n^{-i(1-c)}}{i!} \\
&= L\lambda \sum_{i \geq 1} \frac{(e^\gamma n^{c-1})^i}{i!} \\
&\leq (80\mu + 1) e\mu \left(\exp(e^\gamma n^{c-1}) - 1\right) \\
&\leq 81 e \mu^2 \left(\exp\left(\frac{n^{2c-1}\frac{s}{4}}{81e}\right) - 1\right) \\
&\leq 81 e n^{1-2c} \frac{n^{2c-1}\frac{s}{4}}{81e}(1 + o(1)) \\
&= \frac{s}{4}(1 + o(1)).
\end{aligned} \tag{28}$$

Therefore, we have

$$E\left[e^{\gamma(Z_{t+1}-Z_t)}\right] \leq o(1) + \left(1 - \frac{3s}{4}\right) + \frac{s}{4} + o(1) = 1 - \frac{s}{2} + o(1) \leq 1 - \frac{s}{3},$$

if n is large enough. We also compute

$$D := E\left[e^{\gamma(Z_{t+1}-a(n))} \mid Z_t \leq a(n)\right] \leq 1 + \sum_{i \geq 1} e^{\gamma i} \Pr[Z_{t+1} - a = i \mid Z_t \leq a(n)].$$

The probabilities in the sum are maximized when $Z_t = a(n)$, therefore, we can bound it as in (28) with $\frac{s}{4}(1 + o(1)) \leq \frac{s}{2}$. Hence,

$$D \leq 1 + \frac{s}{2}.$$

Finally, by the negative drift theorem (Theorem 1), the probability that the minimal time T for which $Z_T = n$ is less than t is

$$\Pr[T \leq t] \leq tD\frac{3}{s}e^{-\gamma(b(n)-a(n))} \leq t\left(1+\frac{s}{2}\right)\frac{3}{s}e^{\gamma n^c} = \Theta(tn^{cn^c}),$$

which is sub-constant for any polynomial t. Therefore, the expected runtime is super-polynomial. □

2.3.3 Polynomial Runtime on the Threshold for the Large Population Sizes

In this section we reveal a tighter threshold for the parent and offspring population sizes of the (μ,λ) EA that guarantees a polynomial runtime for the optimization of OneMax. We consider $\lambda \geq e\mu$ and $\mu = \omega(n^{\frac{2}{3}}\log^4(n))$. Such relatively large values of μ give us a high concentration of several random variables such as the number of the individuals on the top level. This concentration turns out to be enough for even small drifts to play a significant role.

In this subsection we define level i as the set of all bit strings of length n with exactly i one-bits. We denote by $X_t(f)$ the number of individuals in P_t of fitness exactly f and by $Y_t(f)$ the number of individuals in population P_t that have a fitness strictly greater than f.

We say that the *current level* is f at generation t if there exists $t_0 < t$ such that $X_{t_0}(f)+Y_{t_0}(f) \geq \frac{\mu}{2}$ and for all $\tau \in [t_0..t]$ we have $X_\tau(f)+Y_\tau(f) \geq \frac{\mu}{4}$ and $Y_\tau(f) < \frac{\mu}{2}$. In other words, it is the highest fitness level such that once there were at least $\frac{\mu}{2}$ individuals on this level or above, and since then this number of individuals has not fallen below $\frac{\mu}{4}$.

The current level f can change in one of the two events. (i) There are less than $\frac{\mu}{4}$ individuals with fitness at least f in the population (then we say that *the algorithm loses a level*) or (ii) there are at least $\frac{\mu}{2}$ individuals with fitness more than f in the population (then we say that *the algorithm gains a level*).

For brevity we define $X_t \coloneqq X_t(f)$ and $Y_t \coloneqq Y_t(f)$, if the current level is f. The main result of this section is the following theorem.

Theorem 19. *If $\lambda \geq e\mu$ and $\mu \geq n^{\frac{2}{3}}\ln^4(n)$ and $\frac{\lambda}{\mu}$ is at most polynomial in n then the expected number of generations of the (μ,λ) EA on the OneMax function is at most $O(n\log(n))$.*

To prove Theorem 19, we split the runtime into two phases. In the first phase, the current level is at most $\frac{n}{3}$. In the second phase the the current level f is greater than $\frac{n}{3}$. We first analyze these two phases separately and then we come up with the proof of Theorem 19.

Lemma 48. *The expected number of generations that the (μ,λ) EA spends in first phase is $O(n)$.*

Proof. Let the current level be $f \leq \frac{n}{3}$ at generation $t = 0$. Then we have $X_0 \geq \frac{\mu}{2}$.

We have $Y_1 \succeq \min\left\{\mu, \mathrm{Bin}(\lambda, \frac{2X_0}{3e\mu})\right\}$, since we create an offspring with fitness at least $f+1$ when we select an offspring with witness f (with probability $\frac{X_0}{\mu}$) and flip only one wrong bit in it (with probability at least $\frac{2}{3e}$). By Chernoff bounds we have the probability that $Y_1 < \frac{\mu}{3}(1-\mu^{-\frac{1}{3}})$ is at most $\exp(-\frac{\mu^{\frac{1}{3}}}{6})$.

At the same time we have $X_1 \succeq \min\{\mu, \mathrm{Bin}(\lambda, \frac{X_0}{e\mu})\}$ and by Chernoff bounds we have the probability that $X_1 < X_0(1-\mu^{-\frac{1}{3}})$ is at most $\exp(-\frac{\mu^{\frac{1}{3}}}{4})$.

Given that $Y_1 \geq \frac{\mu}{3}(1-\mu^{-\frac{1}{3}})$ and $X_1 \geq \frac{\mu}{2}(1-\mu^{-\frac{1}{3}})$ we have $Y_2 \succeq \min\{\mu, \mathrm{Bin}(\lambda, \frac{2X_1}{3e\mu} + \frac{Y_1}{e\mu})\}$. Therefore, by Chernoff bounds we have

$$\begin{aligned}\Pr\left[Y_2 < \frac{\mu}{2}\right] &\leq \Pr\left[Y_2 < \left(Y_1 + \frac{2X_1}{3}\right)(1-\mu^{-\frac{1}{3}})\right] \\ &\leq \exp\left(-\frac{\frac{2\mu}{3}(1-\mu^{-\frac{1}{3}})}{2\mu^{\frac{2}{3}}}\right) \leq \exp\left(-\frac{\mu^{\frac{1}{3}}}{3}\right),\end{aligned}$$

if $\mu \geq 3$. By union bound, the probability that the algorithm does not gain a level in two iterations is at most $\exp(-\frac{\mu^{\frac{1}{3}}}{6}) + \exp(-\frac{\mu^{\frac{1}{3}}}{4}) + \exp(-\frac{\mu^{\frac{1}{3}}}{3}) \leq 3\exp(-\frac{\mu^{\frac{1}{3}}}{6})$.

The probability that the algorithm reaches current level $\frac{n}{3}$ in not more than $\frac{2n}{3}$ iterations is at least $(1-3\exp(-\frac{\mu^{\frac{1}{3}}}{6}))^{\frac{2n}{3}} = 1-o(1)$. If the algorithm does not reach current level $\frac{n}{3}$ in this number of iterations, in the worst case its current level is zero, and it starts another attempt. The probability that the algorithm needs another attempt is $o(1)$, so if n is large enough the expected number of such attempts is at most 2. This results that the expected number of generations of the first phase is at most $\frac{4n}{3} = O(n)$. □

To analyze the expected runtime of the second phase, we regard in details the behavior of the algorithm on the current level f. We show that the algorithm is not likely to lose the current level in a sufficiently long time. At the same time, it creates enough offspring with fitness at least $f+1$ to fill the upper level in expected number of $O(\frac{n}{n-f})$ generations. The first observation leads us to the following lemma.

Lemma 49. *Let the current level at generation $t_0 = 0$ be $f \geq \frac{n}{3}$ and $\mu = n^{\frac{2}{3}}h(n)$, where $h(n) \geq \ln^4(n)$. If $X_0 \geq \frac{\mu}{2}$ and $\lambda \geq e\mu$ and $\frac{\lambda}{\mu}$ is at most polynomial in n and n is large enough then for any $t \in \mathbb{N}$ we have*

$$\Pr\left[\forall \tau \in [t_0..t] \Rightarrow X_\tau \geq \frac{\mu}{4}\right] \geq \left(1-\frac{2}{n^3}\right)^t.$$

The proof of Lemma 49 is based on the following observation. X_t performs an unbiased random walk with steps of size $O(\sqrt{X_t})$. For this reason the expected number of generations before $X_t \leq \frac{\mu}{4}$ is linear in μ. However, while $X_t \geq \frac{\mu}{4}$ we have a positive drift of order $\Theta(\frac{\mu}{n})$ for Y_t, that makes us reach $Y_t = \omega(\sqrt{\mu})$ in $o(\mu)$ iterations with high probability.

Once $Y_t = \omega(\sqrt{\mu})$, it is not likely to decrease by a factor more than 2 in $\sqrt{\mu}$ iterations, since it preforms a random walk of the same manner as X_t did. At the same time such great Y_t creates an influx of individuals of fitness f that is of greater order than the steps made by X_t. This is enough for X_t to become at least $\frac{\mu}{2}$ again before Y_t becomes too small. This regular refilling of level f does not let X_t fall below $\frac{\mu}{4}$ for long enough.

Proof of Lemma 49. We split the runtime of the algorithm while its current level is f into cycles. Each cycle can be either *successful*, *unsuccessful* or *totally unsuccessful*. After a *successful* cycle the algorithm has at least $\frac{\mu}{2}$ individuals of fitness exactly f in the population. After an *unsuccessful* cycle that started with m individuals of fitness f in the population, there are at least $m - 2\Delta_\mu$

individuals of fitness f in the population, where $\Delta_\mu := n^{\frac{2}{3}}h^{\frac{3}{4}}(n)$. We pessimistically assume that an uninterrupted series of $\frac{\mu}{8\Delta_\mu} = \frac{h^{\frac{1}{4}}(n)}{8}$ unsuccessful cycles results into a level loss, since it might decrease the number of individuals at level f by $\frac{\mu}{4}$. A *totally unsuccessful* cycle is a cycle that is neither successful, nor unsuccessful. After a totally unsuccessful cycle we pessimistically assume that the algorithm loses a level. We aim to show that the event of a level loss is not likely to happen for long enough.

Each cycle is split into two phases. Consider some cycle that starts at generation τ_0. To shorten the notation assume that $\tau_0 = 0$, however it does not mean that we regard only the first cycle. The first phase of the cycle terminates after τ_1 generations, that is, at the first generation such that either $X_{\tau_1} < X_0 - \Delta_\mu$ or $Y_{\tau_1} \geq \mu_0$, where $\mu_0 := n^{\frac{1}{3}}h(n)$. If at the end of the first phase we have $X_{\tau_1} < X_0 - \Delta_\mu$ then the cycle is terminated and we consider it as either unsuccessful or totally unsuccessful if $X_{\tau_1} < X_0 - 2\Delta_\mu$. Otherwise, the cycle enters the second phase, which starts with at least μ_0 individuals of fitness greater than f.

The second phase (and also the cycle) ends after τ_2 more generations, where τ_2 is the first integer such that $X_{\tau_1+\tau_2} \geq \frac{\mu}{2}$ or $X_{\tau_1+\tau_2} < X_{\tau_1}$ or $Y_{\tau_1+\tau_2} < \frac{\mu_0}{2}$. If $X_{\tau_1+\tau_2} \geq \frac{\mu}{2}$, then the cycle is successful, otherwise it is either unsuccessful or totally unsuccessful if $X_{\tau_1+\tau_2} < X_{\tau_1}$.

Now we consider each phase in details to show that the probability that the cycle is successful is sufficiently high.

The First Phase. The probability that the cycle is neither unsuccessful nor totally unsuccessful after the first phase is at least the probability that for $\tau_1^* := n^{\frac{2}{3}}h^{\frac{1}{4}}(n)$ the number Y_t of individuals on the upper levels reaches μ_0 in less than τ_1^* generations while the number X_t of individuals on the current level X_t does not go below $X_0 - \Delta_\mu$ until generation τ_1^*. By Lemma 27 we have the probability that $X_t \leq X_0 - \Delta_\mu$ for some $t \leq \tau_1^*$ is at most $\frac{\tau_1^* X_0}{\Delta_\mu^2} \leq \frac{1}{h^{\frac{1}{4}}(n)}$.

To estimate the expected runtime before Y_t becomes at least μ_0 we apply Lemma 29 to Y_t. For this reason we note that $Y_{t+1} \succeq \min\{\mu, \mathrm{Bin}(\lambda, \frac{Y_t}{e\mu} + \frac{X_t}{en\mu})\}$, since we can obtain individuals in the upper levels either by copying one of Y_t individuals with probability at least $\frac{1}{e}$ or by creating a superior offspring from one of X_t individuals with probability at least $\frac{(n-f)}{en} \geq \frac{1}{en}$. Since during the cycle we have X_t at least $\frac{\mu}{4}$, we have $\Delta_{\min} \geq \frac{\mu}{4n}$. By taking $X' = \mu_0$ (which is greater than $\max\{48, 18\ln\frac{\lambda}{\Delta_{\min}}\}$ if n is large enough) and applying Lemma 29 we obtain that the expected runtime before Y_t exceeds μ_0 for the first time is at most $\max\{\frac{4\mu_0 n}{\mu}, 24\} = 4n^{\frac{2}{3}}$, if n is large enough. By Markov's inequality the probability that this time exceeds τ_1^* is at most

$$\frac{4n^{\frac{2}{3}}}{\tau_1^*} = \frac{4}{h^{\frac{1}{4}}(n)}.$$

So as an intermediate result we have the probability that the cycle is neither unsuccessful, nor totally unsuccessful after the first phase is at least $1 - \frac{1}{h^{\frac{1}{4}}(n)} - \frac{4}{h^{\frac{1}{4}}(n)} = 1 - \frac{5}{h^{\frac{1}{4}}(n)}$.

The Second Phase. Proceeding to the second phase of the cycle, we notice that it starts with $X_{\tau_1} \geq \frac{\mu}{4}$ and $Y_{\tau_1} \geq \mu_0$ and the cycle is successful if after the second phase we have $X_{\tau_1+\tau_2} \geq \frac{\mu}{2}$.

The probability that the cycle succeeds after the second phase is at least the probability that for $\tau_2^* := n^{1/3}h^{1/3}(n)$ we have $Y_t \geq \frac{\mu_0}{2}$ for all $t \in [\tau_1..\tau_1 + \tau_2^*]$ and there exists some $t \leq \tau_2^*$ such that $X_{\tau_1+t} \geq \frac{\mu}{2}$ and X_t does not decrease for $t \in [\tau_1, \tau_1 + \tau_2^*]$.

By Lemma 27 the probability that $Y_t < \frac{\mu_0}{2}$ for some $t < \tau_1 + \tau_2^*$ is at most $\frac{4\tau_2^* Y_{\tau_1}}{\mu_0^2} \leq \frac{4}{h^{1/2}(n)}$.

By the application of Lemma 29 to $X_{t+1} \sim \text{Bin}(\lambda, \frac{X_t}{e\mu} + \frac{fY_t}{e\mu})$ we have the expected runtime before X_t becomes more than $\frac{\mu}{2}$ is at most $\frac{4\mu/2-2\mu/4}{\frac{\mu_0}{6}} = 9n^{\frac{1}{3}}$. By Markov's inequality we have that the probability that X_t does not become greater than $\frac{\mu}{2}$ in less than τ_2^* generations is at most $\frac{9n^{\frac{1}{3}}}{\tau_2^*} = \frac{9}{h^{\frac{1}{2}}(n)}$.

Finally, by Chernoff bounds the probability that X_t decreases in one generation during the second phase is at most $\exp(-\frac{Y_t^2}{9E[X_t]}) \leq \exp(-\frac{\mu_0^2}{36\mu}) \leq \exp(-\frac{h(n)}{36})$. The probability that X_t does not decrease during τ_2^* generations is at least $(1-\exp(-\frac{h(n)}{36}))^{n^{\frac{1}{3}}h^{\frac{1}{2}}(n)}$. Since we have $h(n) \geq \ln^4(n)$, if n is large enough this probability is at least $1 - \frac{1}{h^{\frac{1}{2}}(n)}$.

Summing up, the probability that the cycle is not successful after the second phase is at most $\frac{4}{h^{\frac{1}{2}}(n)} + \frac{9}{h^{\frac{1}{2}}(n)} + \frac{1}{h^{\frac{1}{2}}(n)} + \leq \frac{15}{h^{\frac{1}{2}}(n)}$. If we also take into account the probability not to be unsuccessful after the first phase, we estimate the probability of a successful cycle p_s as

$$p_s \geq 1 - \frac{5}{h^{\frac{1}{4}}(n)} - \frac{15}{h^{\frac{1}{2}}(n)} \geq 1 - \frac{6}{h^{\frac{1}{4}}(n)},$$

if n is large enough.

Losing a Level. The cycle is totally unsuccessful in two cases. The first case is when in the end of the first phase we have $X_{\tau_1} < X_0 - 2\Delta_\mu$. Notice that for this to happen we need $X_{\tau_1-1} - X_{\tau_1} > \Delta_\mu$, and by Chernoff bounds the probability of this is at most $\exp(-n^{\frac{2}{3}}h^{\frac{1}{2}}(n))$. The second case is when in the end of the second phase we have $X_{\tau_1+\tau_2} < X_{\tau_1} - \Delta_\mu$. The probability of this event is at most the probability that at the last generation of the second phase X_t decreased, that is, at most $\exp(-\frac{h(n)}{9})$. Hence, the probability p_{tu} of a totally unsuccessful cycle is at most

$$\begin{aligned} p_{tu} &\leq \exp(-n^{\frac{2}{3}}h^{\frac{1}{2}}(n)) + \exp\left(-\frac{h(n)}{9}\right) \\ &\leq \exp(-h^{\frac{1}{2}}(n)) \leq \exp(-\ln^2(n)) \leq \frac{1}{n^3}, \end{aligned}$$

if $n > e^3$, since we assume that $h(n) \geq \ln^4(n)$.

Except a totally unsuccessful cycle we have only one possible way to lose a level, that is a series of $\frac{\mu}{8\Delta_\mu}$ unsuccessful cycles in a row. The probability p_{us} of such series is at most

$$p_{us} \leq (1-p_s)^{\frac{\mu}{8\Delta_\mu}} \leq \left(\frac{6}{h^{\frac{1}{4}}(n)}\right)^{\frac{h^{\frac{1}{4}}(n)}{8}} \leq \frac{1}{n^3},$$

if n is large enough.

Since each cycle takes at least one generation, the probability not to lose a level after t generations is at most $(1-p_{tu}-p_{us})^t \geq (1-\frac{2}{n^3})^t$.

□

Next observation is that before the algorithm loses a level, Y_t has a decent positive drift. This gives us the following lemma.

Lemma 50. *If $\lambda \geq e\mu$ and $\frac{\lambda}{\mu}$ is at most polynomial in n and $\mu = n^{\frac{2}{3}}h(n)$ where $h(n) \geq \ln^4(n)$, then the expected runtime before the (μ, λ) EA either loses or gains a level is at most $\frac{8n}{n-f}$ generations. The probability that this results in a level loss is at most $\frac{10}{n}$.*

Proof. Note that $Y_{t+1} \succeq \min\left\{\mu, \mathrm{Bin}(\lambda, \frac{Y_t}{e\mu} + \frac{(n-f)X_t}{en\mu})\right\}$. Before the algorithm loses a level we have $\frac{X_t}{n} \geq \frac{\mu}{4n}$. By Lemma 29, denoting $\Delta_t := \frac{(n-f)X_t}{n} \geq \frac{(n-f)\mu}{4n}$ and $X' = \frac{\mu}{2}$, we have that the expected runtime before $Y_t \geq X'$ is at most $\frac{4X'}{\Delta_{\min}} = \frac{8n}{n-f}$.

The probability that the algorithm gains a level is at least the probability that before generation $\tau := n^2$ the algorithm has not lost a level and it has gained a level before this generation. By Lemma 49 the probability that it has lost a level is

$$1 - \left(1 - \frac{2}{n^3}\right)^{\tau} = 1 - \left(1 - \frac{2}{n^3}\right)^{n^2} \leq 1 - \exp\left(-\frac{2}{n}\right) \leq \frac{2}{n}.$$

By Markov's inequality the probability that the algorithm does not gain a level in τ generations is at most $\frac{8n}{(n-f)n^2} \leq \frac{8}{n}$.

Therefore, by the union bound the probability that the algorithm loses a level before it gains one is at most $\frac{2}{n} + \frac{8}{n} \leq \frac{10}{n}$. □

Now we are ready to prove Theorem 19.

Proof of Theorem 19. If the algorithm does not lose a level, then the expected number T' of generations before it finds the optimum is at most the expected number of generations spent in the first phase plus the expected number of generations spent in each level of the second phase. By Lemmas 48 and 50 we have

$$E[T'] \leq O(n) + \sum_{f=\frac{n}{3}}^{n-1} \frac{8n}{n-f} = O(n\log(n)).$$

By Lemma 50 the probability not to lose a level before reaching the optimum, which requires to gain at most $\frac{2n}{3}$ fitness levels, is at least $(1 - \frac{10}{n})^{\frac{2n}{3}} \geq e^{-20/3}$, if n is large enough. We pessimistically assume that in the event of a level loss, the algorithm goes back to level zero, hence losing a level is equivalent to a restart of the algorithm. However, the expected number of such restarts is not greater than $e^{20/3}$, so the total expected number of generations of the (μ, λ) EA on the ONEMAX function is $O(n\log(n))$. □

2.4 Conclusion of Chapter 2

In this chapter we proposed two novel methods for the analysis of population-based unary EAs and applied them to the analysis of two standard EAs—the $(\mu + \lambda)$ EA and the (μ, λ) EA. We are optimistic that they can extend the toolbox for the theoretical analysis of EAs and we support our optimism with the results of Sections 2.2 and 2.3, where these methods allowed us to solve problems which are fair to name long-standing.

In Section 2.2 we determined (tight apart from constant factors) the runtime of the $(\mu+\lambda)$ EA on the ONEMAX benchmark problem. This is thus one of the few tight runtime analyses taking into account more than a single parameter ([75, 31] are the other two such works we are aware of).

Not surprisingly for a simple function like ONEMAX, our result does not indicate that it is advantageous to use larger parent or offspring populations. Indeed, it follows from [139, Theorem 6.2] (see [26] for a simplified proof) that for any μ and λ the runtime of the $(\mu+\lambda)$ EA stochastically dominates the runtime of the $(1+1)$ EA with best-of-μ initialization. The runtime difference between the $(1+1)$ EA with best-of-μ initialization and with the usual random initialization is small, roughly an additive $\Theta(\sqrt{n \ln \mu})$ term [114].

While our result does not show an advantage of using larger populations, it does show that using moderate-size populations is not overly costly. For example, as long as $\mu, \lambda = O(\log n)$, the $(\mu+\lambda)$ EA takes $\Theta(n \log n)$ fitness evaluations to find the optimum. This observation could indicate that using such population sizes is generally an interesting idea – we could speculate that there is no harm from using such populations, but there could be other advantages.

In the light of recent other work, our work suggests two directions for further research. In [75], a precise runtime analysis for the $(1+\lambda)$ EA with general mutation rate c/n, c a constant, on the ONEMAX benchmark was conducted. It suggests that the precise mutation rate is important when λ is small, but less decisive when λ is large. It would be interesting to know to what extent this result carries over to the $(\mu+\lambda)$ EA. In [6, 61, 53], it was shown that various dynamic choices of the mutation rate can reduce the runtime of the $(1+\lambda)$ EA on ONEMAX. Again, it would be interesting to see to what extend a similar behavior is true for the $(\mu+\lambda)$ EA.

In Section 2.3, we have analyzed how the (μ,λ) EA optimizes the ONEMAX function when the population sizes are chosen close to the efficiency threshold $\lambda \approx e\mu$. This regime is interesting in that there is no clear negative drift, which strongly prevents approaching the global optimum, and in that there is no clear positive drift, which destroys the ability of comma selection to leave local optima (by creating with high probability a copy of the parent population).

Due to the technical challenges in this regime, this first analysis is not fully conclusive, and in fact, we observe that now also the absolute population size plays a role (more than just the need to be at least polynomial). Our results show in particular that close to the threshold, a polynomial runtime is still possible if the population size is not too small (but $n^{2/3+\varepsilon}$ is enough).

This raises the question (and hope) whether in this regime the (μ,λ) EA can overcome premature convergence when optimizing multi-modal optimization problems. Our upper bound proof suggests that in this regime the population is not quickly concentrated on the best-so-far fitness level, but is spread over more than one level. This could ease leaving such a local optimum. Since the analysis of the (μ,λ) EA on multi-modal problems is again a topic little understood, we cannot answer this question easily, but suggest this as an interesting problem for future research.

Chapter 3 The Methods for Precise Analysis of the Evolutionary Algorithms on Plateaus

In this chapter we aim at making progress on several related subjects—we aim at understanding how evolutionary algorithms optimize non-unimodal fitness functions, what mutation operators to use in such settings, how to analyze the behavior of evolutionary algorithms on large plateaus of constant fitness, and in particular, how to obtain runtime bounds that are precise including the leading constant.

The main result presented in this chapter is a very general analysis of how the simplest mutation-based evolutionary algorithm, the $(1+1)$ EA, optimizes the n-dimensional plateau function with plateau parameter $k \in \mathbb{N}$. The precise result of this analysis lets us also estimate the runtime of the $(\lambda \overset{1:1}{+} \lambda)$ EA with constant population size. We allow the algorithms to use any unbiased mutation operator (including, e.g., one-bit flips, standard bit mutation with an arbitrary mutation rate, or the fast mutation operator) as long as the operator flips exactly one bit with probability $\omega(n^{-\frac{1}{2k-2}})$. This assumption is natural, but also ensures that the algorithm can reach all points on the plateau. Denoting the number of bits flipped in an application of this operator by the random variable α, we prove that the expected optimization time of the $(1+1)$ EA is

$$\frac{n^k}{k!\Pr[1 \leq \alpha \leq k]}(1+o(1)).$$

This result, tight apart from lower order terms only, is remarkable in several respects. It shows that the performance depends very little on the particular mutation operator, only the probability to flip between 1 and k bits has an influence. The absolute runtime is also surprising — it is the size of the plateau times the waiting time until we flip between 1 and k bits.

We also deliver relatively precise estimates of the runtime distribution, which turns to be close to the geometric one with success probability

$$\frac{k!\Pr[1 \leq \alpha \leq k]}{n^k}(1+o(1)).$$

Such precision allows us to extend the runtime results on a population-based algorithm, namely on the $(\lambda \overset{1:1}{+} \lambda)$ EA.

A similar-looking result was obtained in [72], namely that the expected runtime of the $(1+1)$ EA with 1-bit mutation and with standard bit mutation with rate $\frac{1}{n}$ on the needle function is (apart from lower order terms) the size of the plateau times the probability to flip a positive number of bits (which is 1 for 1-bit mutation and $(1-o(1))(1-\frac{1}{e})$ for standard bit mutation with rate $\frac{1}{n}$). Our result is different from that one in that we consider constrained plateaus of arbitrary (constant) radius $k \geq 2$, and more general in that we consider a wide class of unbiased mutation operators. Despite the difference in the plateaus, the expected runtime is surprisingly similar, which is the size of the plateau times the expected number of iterations until we flip between 1 and k bits (where for the needle function we can take $k = n$).

We note that there is a substantial difference between the case $k = n$ and k constant. Since the needle function consists of a plateau containing the whole search space apart from the optimum, the optimization time in this case is just the hitting time of a particular search point when doing an undirected random walk (via repeated mutation) on the hypercube $\{0,1\}^n$.

For PLATEAU$_k$ with constant k, the plateau has a large boundary. More precisely, almost all[4] search points of the plateau lie on its outer boundary and furthermore, all these search points have almost all their neighbors outside the plateau. Hence a large number of iterations (namely almost all) are lost in the sense that the mutation operator generates a search point outside the plateau (and different from the optimum), which is not accepted. Interestingly, as our result shows, the optimization of such restricted plateaus is not necessarily significantly more difficult (relative to the plateau size) than the optimization of the unrestricted needle plateau.

Our precise runtime analysis allows to deduce a number of particular results. For example, when using standard bit mutation, the optimal[5] mutation rate is $\frac{\sqrt[k]{k!}}{n}$, that is, approximately $\frac{k}{en}$. This is by a constant factor less than the optimal rate of $\frac{k}{n}$ for the jump function with jump size k, but again a factor of $\Theta(k)$ larger than the classic recommendation of $\frac{1}{n}$, which is optimal for many unimodal fitness functions. Hence our result confirms that the optimal mutation rates can be significantly higher for non-unimodal fitness functions. While the optimal mutation rates for jump and plateau functions are similar, the effect of using the optimal rate is very different. For jump functions, an $k^{\Theta(k)}$ factor speed-up (compared to the standard recommendation of $\frac{1}{n}$) was observed, here the influence of the mutation operator is much smaller, namely the factor $\Pr[1 \leq \alpha \leq k]$, which is trivially at most 1, but which was assumed to be at least some positive constant. Interestingly, our results imply that the fast mutation operator is not more effective than other unbiased mutation operators, even though it was proven to be significantly more effective for jump functions and it has shown good results in some practical problems [105].

So one structural finding, which we believe to be true for larger classes of problems and which fits to the result [72] for needle functions, is that the mutation rate, and more generally, the particular mutation operator which is used, is less important while the evolutionary algorithm is traversing a plateau of constant fitness.

The main technical novelty in this chapter is that we model the optimization process via two different Markov chains describing the random walk on the plateau, namely the chain defined on the $\Theta(n^k)$ elements of the plateau (plus the optimum) and the chain obtained from aggregating these into the total mass on the Hamming levels. Due to the symmetry of the process, one could believe that it suffices to regard only the level chain. The chain defined on the elements, however, has some nice features which the level chain is missing, among others, a symmetric transition matrix (because for any two search points x and y on the plateau, the probability of going from x to y is the same as the probability of going from y to x). For this reason, we find it fruitful to switch between the two chains. Exploiting the interplay between the two chains and using classic methods from linear algebra, we find the exact expression for the expected runtime.

The most valuable insight given by this approach is that the mixing of the probability mass over the plateau is very fast. More precisely, we show that independently of the first position on the plateau, in slightly more than $\Theta(\sqrt{n}\log(n))$ iterations we are almost equally likely to be at any point of the plateau. A similar mixing argument was used to prove the upper bound on the runtime of the $(1+1)$ EA on the LEADINGONES with strong prior noise in [126]. There, however, only an exponential mixing time was shown, although the author conjectures that it should be polynomial. Our analysis based on the interplay of two Markov chains is problem-specific (e.g., we base our arguments on the symmetry of the plateau), but we are optimistic that

[4] in the usual asymptotic sense, that is, meaning all but a lower order fraction

[5] We call a mutation rate optimal when it delivers the expected runtime that differs from the truly optimal one at most by lower order terms, that is, e.g. a factor of $(1 \pm o(1))$.

the observed behavior of a small mixing time can be also seen on other plateaus which are not too easy to leave.

The rest of the chapter has the following structure. In Section 3.1 we list the mathematical means that are used in our analysis. In Section 3.2 we introduce the central tool of our analysis — the two Markov chains, show their properties and the connection between the two chains. In Section 3.3 we prove the main result of this chapter, which is, the precise runtime of the $(1+1)$ EA on the $\textsc{Plateau}_k$ function for constant k. The corollaries from the main result, which are, the precise runtime of different variants of the $(1+1)$ EA, are shown in Section 3.4. Finally, we summarize the results in Section 3.6.

3.1 Tools from Linear Algebra

In this section we briefly review the terms, tools and facts from the linear algebra that we use in this chapter. Since they are not used in other chapters, we did not put them into Section 1.8.

Given the square matrix A, the vector x is called the *left eigenvector* of the matrix A if $xA = \lambda x$ for some $\lambda \in \mathbb{C}$. In this situation, λ is called *eigenvalue* of the matrix A. The vector x is called *right eigenvector* if $Ax = \lambda x$ for some $\lambda \in \mathbb{C}$. Since in this chapter we regard only left eigenvectors, we call them just *eigenvectors*.

The *spectrum* of a matrix is the set of all its eigenvalues. If a matrix has size $n \times n$, then the number of its eigenvalues is not greater than n. For each eigenvalue there exists a corresponding *eigenspace*, that is, the linear span of all the eigenvectors that correspond to the eigenvalue.

The only point shared by any two eigenspaces that correspond to two different eigenvalues is 0^n.

The *characteristic polynomial* $\chi(\lambda)$ of matrix A is the function of λ that is defined as the determinant of the matrix $A - \lambda I$, where I is the identity matrix. The set of roots of the characteristic polynomial equals the spectrum of the matrix A.

The standard inner product of the vectors $x = (x_0, \ldots, x_{n-1})$ and $y = (y_0, \ldots, y_{n-1})$ is a scalar value defined by $\langle x, y \rangle = \sum_{i=0}^{n-1} x_i y_i$. The two vectors are *orthogonal* if their inner product is zero.

For every *diagonalizable* matrix A of size $n \times n$ there exists a set $\{e^i\}_{i=0}^{n-1}$ of eigenvectors that form a *basis* of $\mathbb{R}^n$. A basis is called *orthogonal* when all pairs of the basis vectors are orthogonal. A matrix $A = (a_i^j)$ is *symmetric* if for every i and j we have $a_i^j = a_j^i$.

We use the following two properties of symmetric matrices.

Lemma 51. *All eigenvalues of a symmetric matrix are real.*

Lemma 52. *Two eigenvectors of a symmetric matrix that correspond to different eigenvalues are orthogonal. Also every symmetric matrix of size $n \times n$ is diagonalizable, which meas that there exists an orthogonal basis of $\mathbb{R}^n$ which consist of eigenvectors of this matrix.*

In this chapter we also encounter *irreducible* matrices. Among the several definitions, the following is the easiest to check for the non-negative matrices considered in this chapter. For each non-negative matrix A of size $n \times n$ we can build a directed graph G_A by taking an empty

graph on n vertices and adding an edge from vertex i to vertex j for each positive component a_i^j of A. Then a matrix A is *irreducible* if and only if graph G_A is strongly connected.

For example, the transition matrix of an irreducible Markov chain (a chain such that each state is reachable from each other state) is irreducible.

A crucial role in this chapter is played by the Perron-Frobenius theorem [104]. This theorem gives a series of properties of the irreducible matrices, among them we use the following four.

Theorem 20 (Perron-Frobenius)**.** *Any irreducible non-negative matrix A has the following properties.*

- *The largest eigenvalue λ_0 of A lies between the minimal and the maximal row sum of A.*
- *For every eigenvalue λ of A different from the largest eigenvalue λ_0 we have $|\lambda| < \lambda_0$.*
- *The largest eigenvalue of A has a one-dimensional eigenspace.*
- *There exists an eigenvector which corresponds to the largest eigenvalue λ_0 all components of which are strictly positive.*

When talking about vector norms, we use the following notation. For any $p \in (0, +\infty)$ and any vector $x \in \mathbb{R}^n$, we let

$$\|x\|_p = \left(\sum_{j=0}^{n-1} |x_j|^p \right)^{1/p}.$$

In this chapter we use only the Manhattan norm ($p = 1$) and the Euclidean norm ($p = 2$). We use the following property of these norms.

Lemma 53. *For all $x \in \mathbb{R}^n$ we have*

$$\|x\|_2 \le \|x\|_1 \le \sqrt{n}\, \|x\|_2 .$$

The following lemma is often called *triangle inequality*

Lemma 54. *For any norm $\|\cdot\|$ and for every x, y and $z = x + y$ we have*

$$\|x\| - \|y\| \le \|z\| \le \|x\| + \|y\|$$

We use the following properties of the Euclidean norm.

Lemma 55. *If vectors $x^1, \ldots, x^n$ are orthogonal, then for any values $a_1, \ldots, a_n \in \mathbb{R}$ we have*

$$\left\| \sum_{i=1}^{n} a_i x^i \right\|_2 \le \max_{i \in [1..n]} |a_i| \left\| \sum_{i=1}^{n} x^i \right\|_2$$

Lemma 56. *If vectors $x^1, \ldots, x^n$ are orthogonal, then for any subset $S \subset [1..n]$ we have*

$$\left\| \sum_{i \in S} x^i \right\|_2 \le \left\| \sum_{i=1}^{n} x^i \right\|_2$$

We also make a use of the orthogonal projection of vectors, which is defined as follows. Suppose we have vector $x \in \mathbb{R}^n$ and it is decomposed into the sum of m orthogonal vectors $\{x^i\}_{i=0}^{m-1}$ where $m \le n$. Then x^i is the *orthogonal projection* of x to the linear span of x^i. To calculate precisely the norm of the projection, we use the following lemma.

Lemma 57. *If x^i is the orthogonal projection of x, then for any norm $\|\cdot\|$ we have*

$$\left\|x^i\right\| = \frac{\left\langle x, \frac{x^i}{\left\|x^i\right\|}\right\rangle}{\left\langle \frac{x^i}{\left\|x^i\right\|}, \frac{x^i}{\left\|x^i\right\|}\right\rangle}.$$

We also encounter the *self-adjoint operators*. An operator $A : \mathbb{R}^n \to \mathbb{R}^n$ is called *self-adjoint* if for all $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^n$ we have $\langle Ax, y\rangle = \langle x, Ay\rangle$, where $\langle \cdot, \cdot \rangle$ stands for the standard inner product. The operator in this space is self-adjoint if and only if its matrix is symmetric. The most important properties of self-adjoint operators are stated in the Hilbert-Schmidt theorem [119]. We use only one of them.

Lemma 58. *For any self-adjoint operator $A : \mathbb{R}^n \to \mathbb{R}^n$ there exists an orthonormal basis of $\mathbb{R}^n$ that consists of the eigenvectors of A.*

3.2 Proposed Analysis Method for Plateaus

We propose a novel method for the analysis of EAs on plateaus, which is based on the analysis of the spectrum of the transient matrices of two different Markov chains. Although at the moment this method is tailored for the PLATEAU$_k$ function, we are optimistic that insights delivered by our analysis are also true for plateaus of an arbitrary shape. By insights we mean the observation that when the $(1+1)$ EA enters a plateau, in a short time it becomes uniformly distributed over the plateau, that is, each point of the plateau is (almost) equally likely to be the current individual of the $(1+1)$ EA. However, to show that it is also what happens on plateaus of an arbitrary shape, the proposed method requires further development.

In this section we introduce the two Markov chains and prove some of their properties which are essential for our analysis.

3.2.1 Two Markov Chains

For the optimization process of our $(1+1)$ EA we first observe that, since the unbiased operator with probability $\omega(n^{-\frac{1}{2k-2}})$ flips exactly one bit, the expected time to reach the plateau is $O(n^{1+\frac{1}{2k-2}} \log n)$. Since the time for leaving the plateau (as shown in this chapter) is $\Omega(n^k)$, we only consider the runtime of the algorithm after it has reached the plateau.

For this runtime analysis on the plateau we consider the plateau in two different ways. The first way is to regard a Markov chain that contains $N+1$ states, where $N = \sum_{i=0}^{k-1} \binom{n}{k-i}$. Each state represents one element of the plateau plus there is one absorbing state for the optimum. Note that $N = \frac{n^k}{k!} + o(n^k)$, since $\binom{n}{j} = \frac{n^j}{j!}(1+o(1))$ for all $j \in [1..k]$. The transition probability from transient state x to any state y is $p_x^y = \Pr[\alpha = d]\binom{n}{d}^{-1}$, where d is the Hamming distance between x and y. This implies that the transition probability from x to y is equal to the transition probability

from y to x for any pair of the transient states. Therefore, the transient matrix[6] is symmetric, which gives us the opportunity to use Lemma 51 and Lemma 52. We call this Markov chain the *individual chain*, denote its transient matrix by Q and call the space of real vectors of dimension N the *individual space*[7], since the current state of the chain defines the current individual of the algorithm.

To define the second Markov chain, we first define the *i-th level* as the set of all search points that have exactly $n-k+i$ one-bits. Then the plateau is the union of levels 0 to $k-1$ and the optimum is the only element of level k. Notice that the i-th level contains exactly $\binom{n}{k-i}$ elements (search points). For every $i,j \in [0..k]$ we have that for any element of the i-th level the probability to mutate to the j-th level is the same due to the unbiasedness of the operator. Therefore we can regard a Markov chain of $k+1$ states, where the i-th state ($i \in [0..k]$) represents the elements of the i-th level. State k is an absorbing state. The transition probability from level i to level j is

$$p_i^j = \begin{cases} 0, \text{ if } i = k, j \neq k, \\ \sum_{m=0}^{k-j} \binom{k-i}{j-i+m}\binom{n-k+i}{m}\binom{n}{j-i+2m}^{-1} \Pr[\alpha = j-i+2m], \text{ if } j > i, \\ \sum_{m=0}^{k-i} \binom{k-i}{m}\binom{n-k+i}{i-j+m}\binom{n}{i-j+2m}^{-1} \Pr[\alpha = i-j+2m], \text{ if } j < i \text{ and } i \neq k, \\ 1 - \sum_{m=0, m\neq i}^{k} p_i^m, \text{ if } j = i, \end{cases} \tag{29}$$

where we assume that $n > 2k$ not to complicate the upper limit of sums. This assumption is justified by that we only consider constant k and we estimate the runtime with n tending to infinity. We notice the following useful property of these probabilities.

Lemma 59. *For all $i,j \in [0..k-1]$ we have*

$$\binom{n}{k-i} p_i^j = \binom{n}{k-j} p_j^i.$$

Proof. Let L_s denote level s for all $s \in [0..k-1]$. Let also $p_{x\to L_s}$ denote the probability to get from individual x to any individual in level s. Since for all individuals x in level i the probability $p_{x\to L_j}$ is the same and equal to p_i^j and since there are $\binom{n}{k-i}$ individuals in level i, we have

$$\binom{n}{k-i} p_i^j = \sum_{x\in L_i} p_{x\to L_j} = \sum_{x\in L_i}\sum_{y\in L_j} q_x^y = \sum_{x\in L_i}\sum_{y\in L_j} q_y^x = \sum_{y\in L_j} p_{y\to L_j} = \binom{n}{k-j} p_j^i.$$

□

We observe that the probability to gain ℓ levels is $O(n^{-\ell})$.

Lemma 60. *For all $i \in [0..k-1]$ and $j \in [i+1..k]$, we have $p_i^j = O(n^{-(j-i)})$.*

[6]Recall that the transient matrix is a submatrix of the transition matrix consisting of rows and columns which correspond to transient states.

[7]Note that the dimension of the individual space is equal to the number of transient states of the individual chain, not to the total number of states. Hence, matrix Q defines a linear operator on the individual space.

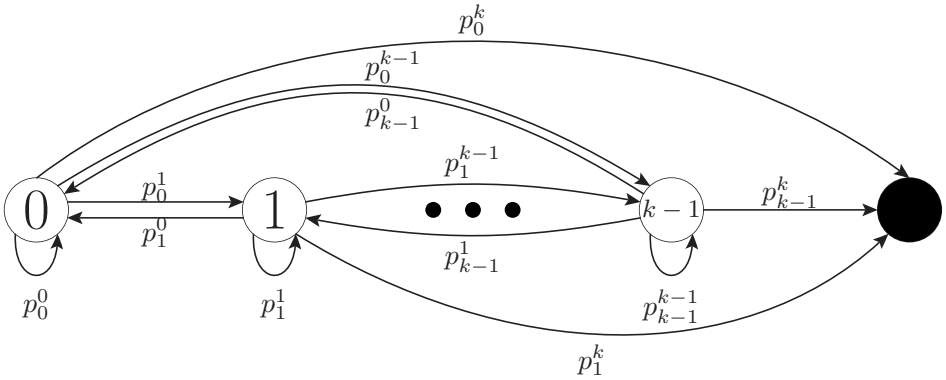


Figure 8 – *Illustration of the level chain. The black circle represents the optimum that is an absorbing state. The states* $[0..k-1]$ *represent the levels of the plateau surrounding the optimum.*

Proof. By Lemma 59 and since $p_j^i \leq 1$ we have

$$p_i^j = \frac{\binom{n}{k-j} p_j^i}{\binom{n}{k-i}} \leq \frac{(n-k+i)!(k-i)!}{(n-k+j)!(k-j)!} = O(n^{-(j-i)}).$$

□

We call this Markov chain the *level chain* and we call the space of real vectors of length k the *level space*[8]. The level chain is illustrated in Fig. 8. The transient matrix P of the leaky chain has a size of $k \times k$. The matrix P (unlike Q) is not symmetric. In our analysis we use the following property of the matrix P.

Lemma 61. *The sum of each row of* P *is* $1 - O(\frac{1}{n})$.

Proof. The sum of the i-th row of P is

$$\sum_{j=0}^{k-1} p_i^j = 1 - p_i^k, \tag{30}$$

since the sum of all the outgoing probabilities for each state in the original Markov chain is one. By Lemma 60 we have $p_i^k = O(n^{-(k-i)}) = O(\frac{1}{n})$. □

There is a natural mapping from the level space to the individual space. Every vector $x = (x_0, \ldots, x_{k-1})$ can be mapped to the vector $\phi(x) = (y_0, \ldots, y_{N-1})$, where $y_m = x_j / \binom{n}{k-j}$, if the m-th element belongs to the j-th level. If x is a distribution over the levels, that is, $x \in [0,1]^k$ and $\|x\|_1 = 1$, then $\phi(x)$ is the distribution over the elements of the plateau which is uniform on the levels and which has the same total mass on each level as x. This mapping has several useful properties.

Lemma 62. ϕ *is linear, that is, we have* $\phi(\alpha x + \beta y) = \alpha\phi(x) + \beta\phi(y)$ *for all* $x, y \in \mathbb{R}^k$ *and all* $\alpha, \beta \in \mathbb{R}$.

[8] As well as for the individual space, the dimension of the level space is equal to the number of the transient states of the level chain and matrix P defines a linear operator on this space.

This property follows directly from the definition of ϕ.

Lemma 63. *For all $x \in \mathbb{R}^k$ we have $\phi(xP) = \phi(x)Q$.*

Proof. In informal words, this property holds because both matrices P and Q represent the same operator, but in different spaces. Thus, the result of applying this operator to some vector and then switching the space is the same as performing these two actions in a reversed order.

For the formal proof, recall that *level i* is the set of all individuals in distance $(k-i)$ from the optimum. In the same manner as in the proof of Lemma 59 we use the fact that for any individual m in level j we have

$$p_j^i = \sum_{\ell \in \text{level } i} q_m^\ell,$$

where q_m^ℓ is the element of matrix Q, that is, the probability to obtain individual ℓ from individual m. From this and from the definition of ϕ we calculate the m-th element of $\phi(xP)$, assuming that individual m belongs to level j.

$$(\phi(xP))_m = \frac{(xP)_j}{\binom{n}{k-j}} = \frac{\sum_{i=0}^{k-1} x_i p_i^j}{\binom{n}{k-j}}.$$

By Lemma 59 we have $\binom{n}{k-i} p_i^j = \binom{n}{k-j} p_j^i$. Therefore,

$$(\phi(xP))_m = \sum_{i=0}^{k-1} \frac{x_i}{\binom{n}{k-i}} p_j^i = \sum_{i=0}^{k-1} \frac{x_i}{\binom{n}{k-i}} \sum_{\ell \in \text{level } i} q_m^\ell.$$

Recall that $q_m^\ell = q_\ell^m$ for all $\ell, m \in [0..N-1]$. Hence, we have

$$(\phi(xP))_m = \sum_{i=0}^{k-1} \frac{x_i}{\binom{n}{k-i}} \sum_{\ell \in \text{level } i} q_\ell^m = \sum_{\ell=0}^{N-1} (\phi(x))_\ell \cdot q_\ell^m = (\phi(x)Q)_m.$$

□

Lemma 64. *The spectrum $\sigma(P)$ of the matrix P is a subset of the spectrum $\sigma(Q)$ of the matrix Q. For any eigenvector x of the matrix P the vector $\phi(x)$ is an eigenvector of Q.*

Proof. From Lemma 62 and Lemma 63 it follows that if x is an eigenvector of P, then $\phi(x)$ is an eigenvector of Q with the same eigenvalue. Thus, every eigenvalue of P is an eigenvalue of Q. □

Lemma 65. *For all $x \in \mathbb{R}^k$, the Manhattan norm is invariant under ϕ, that is, $\|x\|_1 = \|\phi(x)\|_1$.*

This follows from the fact that all components of $\phi(x)$ that are from the same level have the same sign. Notice that an analogous property does not hold for the Euclidean norm $\|\cdot\|_2$.

Although the two Markov chains represent the same process and each of them contains all information about it, in our analysis we need to use both of them simultaneously. We do not really work with the whole individual space, but only with its subspace $\phi(S)$, where S is the level space. Hence, it is natural to use the terms of the level space to simplify the computations and make them easier to understand. On the other hand, we cannot prove some essential facts about the operator $\mathcal{P}$ represented by the matrix P, e.g., that there exists a basis of the level space

which consists of the eigenvectors of $\mathcal{P}$ (see Lemma 67). To prove them we have to switch to the individual space (or more precisely, to its subspace $\phi(S)$) and use the properties of the self-adjoint operator $\mathcal{Q}$ represented by the symmetric matrix Q. Therefore, both chains and their transient matrices are indispensable in our analysis.

3.2.2 The Spectrum of the Transient Matrix

The main result of this section is the following analysis of the eigenvalues of P, which builds on the interplay between the two Markov chains.

Lemma 66. *Let P be the transient matrix of the level chain. Then the following three properties hold.*

1) *All eigenvalues of P are real.*
2) *The largest eigenvalue λ_0 of P satisfies $\lambda_0 = 1 - O(1/n)$.*
3) *Let $\Pr[\alpha = 1] > 0$ and $\Pr[\alpha = 1] = \omega(1/\sqrt[k-1]{n})$. Then with $c := \Pr[\alpha = 1]$ and with $\varepsilon := \frac{c^{k-1}}{(k-1)2^k}$ any other eigenvalue $\lambda' \neq \lambda_0$ of P satisfies $|\lambda'| < 1 - \varepsilon$.*

Proof. The fact that the eigenvalues are real follows from the facts that by Lemma 64 the spectrum of P is a subset of the spectrum of Q and that by Lemma 51 all eigenvalues of the symmetric matrix Q are real.

The largest eigenvalue λ_0 of P is bounded by the minimal and the maximal row sum of P (see Theorem 20), which are both $1 - O(1/n)$ by Lemma 61.

It remains to show that the absolute values of all other eigenvalues are less than $1 - \varepsilon$ for $\varepsilon = \frac{c^{k-1}}{(k-1)2^k}$, which requires more work. To prove this statement we perform a precise analysis of the characteristic polynomial of P.

Recall that the spectrum of P is the set of the roots of its characteristic polynomial

$$\chi_P(\lambda) = \det(P - \lambda I) = \sum_{\sigma \in S_k} \operatorname{sgn}(\sigma) \prod_{i=0}^{k-1} (P - \lambda I)_{i,\sigma(i)},$$

where S_k is the set of all permutations of the set $[0..k-1]$ and $\operatorname{sgn}(\sigma)$ denotes the *signature* of permutation σ (that is, $+1$ if it can be obtained from the identity permutation in even number of element swaps, and -1 otherwise). Note that for all permutations except the identity the product in the sum contains at least one factor $(P - \lambda I)_{i,j}$ with $j > i$ and this element satisfies $(P - \lambda I)_{i,j} = p_i^j = O(1/n)$ by Lemma 60. The other factors of the product are either $p_{i'}^{j'}$ or $(p_{i'}^{i'} - \lambda)$ for some i', j', therefore every product where σ is not the identity is a polynomial in λ with coefficients which are $O(1/n)$. Thus, the characteristic polynomial can be written as

$$\chi_P(\lambda) = \prod_{i=0}^{k-1} (p_i^i - \lambda) + \beta(\lambda), \tag{31}$$

where $\beta(\lambda)$ is some polynomial in λ with coefficients that are all $O(1/n)$. For this reason the derivative $\beta'(\lambda)$ will also be $O(1/n)$ for all $\lambda \in [-1, 1]$, where we recall that all asymptotics are for $n \to \infty$ (and, e.g., not for any limit behavior of λ).

To prove that for all eigenvalues $\lambda' \neq \lambda_0$ we have $\lambda' < 1-\varepsilon$ we need to prove that there is no more than one root of the characteristic polynomial in $[1-\varepsilon, 1]$. To do so it suffices to prove that $\chi_P(\lambda)$ is strictly monotonic in this segment.

Consider $\lambda \geq 1-\varepsilon$. This implies that $\lambda \geq 1 - \frac{c}{2}$. For every $i \neq 0$ we have $p_i^i \leq 1 - \Pr[\alpha = 1] = 1 - c$. Thus, for every $i \neq 0$ and any $\lambda \geq 1-\varepsilon$ we have

$$(p_i^i - \lambda) \leq -c/2. \tag{32}$$

By (31), the derivative of $\chi_P(\lambda)$ can be written as

$$\chi_P'(\lambda) = (p_0^0 - \lambda)\left(\prod_{i=1}^{k-1}(p_i^i - \lambda)\right)' - \prod_{i=1}^{k-1}(p_i^i - \lambda) + \beta'(\lambda). \tag{33}$$

Recall that $\varepsilon = \frac{c^{k-1}}{(k-1)2^k}$. For all $\lambda \geq 1-\varepsilon$ we have

$$p_0^0 - \lambda \leq 1 - \left(1 - \frac{c^{k-1}}{(k-1)2^k}\right) = \frac{c^{k-1}}{(k-1)2^k},$$

$$\left|\left(\prod_{i=1}^{k-1}(p_i^i - \lambda)\right)'\right| = \left|-\sum_{i=1}^{k-1}\prod_{\substack{j\in[1..k-1]\\ j\neq i}}(p_j^j - \lambda)\right| \leq (k-1),$$

$$\left|\prod_{i=1}^{k-1}(p_i^i - \lambda)\right| \geq \frac{c^{k-1}}{2^{k-1}},$$

where the last inequality follows from (32). Furthermore, from (32) it also follows that for $i \neq 0$ we have $(p_i^i - \lambda) < 0$. Thus,

$$\operatorname{sign}\left(\left(\prod_{i=1}^{k-1}(p_i^i - \lambda)\right)'\right) = \operatorname{sign}\left(-\sum_{i=1}^{k-1}\prod_{\substack{j\in[1..k-1]\\ j\neq i}}(p_j^j - \lambda)\right) = (-1)^{k-1}. \tag{34}$$

Consequently, we have two cases.

Case 1: When $p_0^0 - \lambda \geq 0$, we have

$$\begin{aligned}
&\left|(p_0^0 - \lambda)\left(\prod_{i=1}^{k-1}(p_i^i - \lambda)\right)' - \prod_{i=1}^{k-1}(p_i^i - \lambda)\right| \\
&\quad\geq \left|\prod_{i=1}^{k-1}(p_i^i - \lambda)\right| - \left|(p_0^0 - \lambda)\left(\prod_{i=1}^{k-1}(p_i^i - \lambda)\right)'\right| \\
&\quad\geq \left|\frac{c}{2}\right|^{k-1} - \frac{c^{k-1}}{(k-1)2^k}\sum_{i=1}^{k-1}\left|\prod_{\substack{j\in[1..k-1]\\ j\neq i}}(p_j^j - \lambda)\right| \\
&\quad\geq \frac{c^{k-1}}{2^{k-1}} - (k-1)\frac{c^{k-1}}{(k-1)2^k} = \frac{c^{k-1}}{2^k}.
\end{aligned}$$

Hence, we have

$$|\chi'_P(\lambda)| = \frac{c^{k-1}}{2^k} + O(1/n) = \frac{\omega(1/n^{\frac{1}{k-1}})^{k-1}}{2^k} + O(1/n) = \omega(1/n).$$

Case 2: When $p_0^0 - \lambda < 0$, since by (32) $(p_i^i - \lambda) < 0$ for all $i \neq 0$, we have

$$\left|(p_0^0 - \lambda)\left(\prod_{i=1}^{k-1}(p_i^i - \lambda)\right)' - \prod_{i=1}^{k-1}(p_i^i - \lambda)\right| \geq \left|\prod_{i=1}^{k-1}(p_i^i - \lambda)\right| \geq \frac{c^{k-1}}{2^{k-1}}.$$

Therefore,

$$|\chi'_P(\lambda)| = \omega(1/n).$$

For n large enough, this together with (33) and (34) implies that $\chi'_P(\lambda)$ has the same sign as $(-1)^{k-1}$ for every $\lambda \in [1-\varepsilon, 1]$. Thus, there can be only one root of characteristic polynomial in this segment.

To rule out that there is a negative eigenvalue λ with $|\lambda| > 1-\varepsilon$, we notice that for $\lambda < -\frac{1}{2}$ and for every i we have $(p_i^i - \lambda) > \frac{1}{2}$. Therefore, $|\chi_P(\lambda)| > \left(\frac{1}{2}\right)^k - o(1)$, and thus there are no roots that are less than $-\frac{1}{2}$ when n is large enough.

This finally shows that for all eigenvalues $\lambda' \neq \lambda_0$ we have $|\lambda'| < 1 - \min\left(\frac{1}{2}, \frac{c^{k-1}}{(k-1)2^k}\right) = 1 - \frac{c^{k-1}}{(k-1)2^k}$.

□

3.3 Runtime Analysis

In this section, we prove our main result, which determines the runtime of the $(1+1)$ EA on the PLATEAU function.

Theorem 21. *Consider the $(1+1)$ EA using any unbiased mutation operator such that the probability to flip exactly one bit is at least $\Pr[\alpha = 1] = \omega\left(n^{-\frac{1}{2k-2}}\right)$. Let T denote the runtime of this algorithm starting on an arbitrary search point of the plateau of the PLATEAU$_k$ function. Then*

$$E[T] = \frac{n^k}{\Pr[1 \leq \alpha \leq k]k!}(1 + o(1)),$$
$$\Pr[T > t] = (1 \pm o(1))\left(1 - \frac{k!\Pr[1 \leq \alpha \leq k]}{n^k}(1 \pm o(1))\right)^t + r(t),$$

where $|r(t)| \leq \sqrt{N}(1-\varepsilon)^t$, $\varepsilon = \frac{(\Pr[\alpha=1])^{k-1}}{(k-1)2^k}$, and $N = \frac{n^k}{k!}(1 \pm o(1))$. All asymptotic notation refers to $n \to \infty$ and is independent of t.

We start with a few preparatory results. Recall that by Theorem 20 the largest eigenvalue of a positive matrix has a one-dimensional eigenspace. Also this theorem asserts that both left

and right eigenvectors that correspond to the largest eigenvalue have all components with the same sign and they do not have any zero component. Let π^* be such a left eigenvector with positive components for P and let it be normalized in such way that $\|\pi^*\|_1 = 1$. We view π^* as distribution over the levels of the plateau and call it the *conditional stationary distribution of P* since it does not change in one iteration under the condition that the algorithm does not find the optimum. Also let $u = (u_0, \dots, u_{k-1})$ be the probability distribution in the level space such that $\phi(u)$ is the uniform distribution in the individual space. Hence

$$u_i = \binom{n}{k-i} \Big/ \sum_{j=0}^{k-1} \binom{n}{k-j} = \binom{n}{k-i} N^{-1}$$

for all $i \in [0..k-1]$. Our next target is showing that π^* and u are asymptotically equal. For this, we need the following basis of the level space.

Lemma 67. *There exists a basis $\{e^i\}_{i=0}^{k-1}$ of the level space with the following properties.*

1) $\pi^* = e^0$.

2) e^i is an eigenvector of P for all $i \in [0..k-1]$.

3) The $\phi(e^i)$ are orthogonal in the individual space.

Proof. Let S be the level space and S_{ind} be the individual space. Then $\phi(S)$ is a subspace of S_{ind} with $\dim \phi(S) = k$, since the kernel of ϕ is trivial.

Consider the operator $\mathcal{Q}$ that is represented by the matrix Q. It is a self-adjoint operator on S_{ind}, since its matrix is symmetric. Moreover, this operator maps $\phi(S)$ into $\phi(S)$, since for all $x \in S$ by Lemma 63 we have $\mathcal{Q}(\phi(x)) = \phi(x)Q = \phi(xP)$. Therefore, $\mathcal{Q}$ is a self-adjoint operator on $\phi(S)$. Thus, by Lemma 58 there exists an orthonormal basis $f^0, \dots, f^{k-1}$ of $\phi(S)$ that consists of eigenvectors of $\mathcal{Q}$. Let $e^i = \phi^{-1}(f^i)$ for all $i \in [0..k-1]$. By Lemma 64 the e^i are eigenvectors of P. By the linearity of ϕ (Lemma 62) they are linearly independent, hence they form a basis of S.

By assuming that e^0 corresponds to the largest eigenvalue and multiplying e^0 by a suitable scalar, we also satisfy the first property of the lemma. □

We use the basis from Lemma 67 to prove that $\phi(\pi^*)$ is very close to the uniform distribution.

Lemma 68. *If $\Pr[\alpha = 1] = \omega\left(n^{-\frac{1}{2k-2}}\right)$, then for all $j \in [0..k-1]$, we have $\pi^*_j = u_j(1+\gamma_j)$, where $|\gamma_j| \le \gamma$ for some $\gamma = o(1)$.*

Proof. Figure 9 illustrates the relation of the terms used in this proof to make it easier to follow. By Lemma 67, there exist unique $c_0, c_1, \dots, c_{k-1} \in \mathbb{R}$ such that

$$u = \sum_{i=0}^{k-1} c_i e^i.$$

If we transfer this decomposition into the individuals space (using the linearity of ϕ, see Lemma 62), we obtain

$$U = \sum_{i=0}^{k-1} U^i,$$

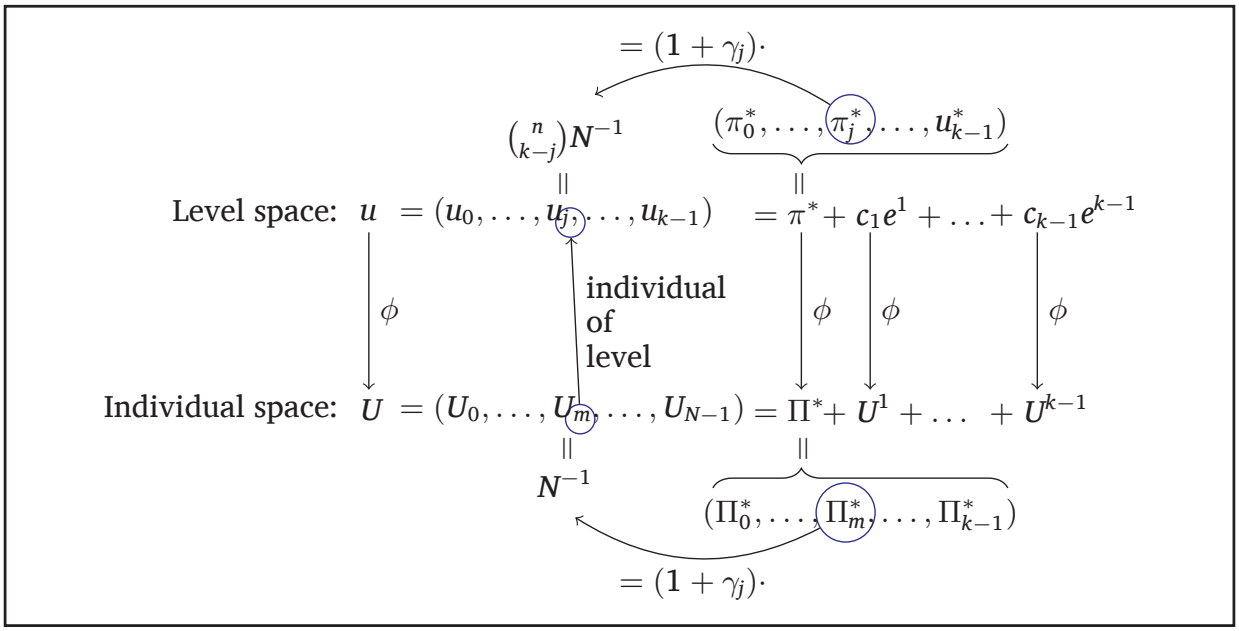


Figure 9 – *Illustration of the terms and their relations used in Lemma 68*

where we define $U \coloneqq \phi(u)$ and $U^i \coloneqq \phi(c_i e^i)$ for all $i \in [0..k-1]$. Note that the vector U describes the uniform distribution in the individuals space and hence all its components are equal to $\frac{1}{N}$. For brevity we also define $\Pi^* \coloneqq \phi(\pi^*)$.

We now aim at finding a useful connection between the components of Π^* and U. Namely, if for all levels $j \in [0..k-1]$ we prove that for any individual m in level j we have $\Pi_m^* = (1+\gamma_j)U_m$ with γ_j that satisfies the conditions of the theorem, we simultaneously prove the same relation for the components of π^* and u.

Recall that π^* is a normalized vector. Thus, by Lemma 65 we have $\|\Pi^*\|_1 = 1$. Recall also that $\pi^* = e^0$ (by the choice of the basis) and therefore, we have $\Pi^* = \frac{U^0}{\|U^0\|_1}$. For all $m \in [0..N-1]$, we have

$$\Pi_m^* = \frac{U_m^0}{\|U^0\|_1}. \tag{35}$$

The m-th component of U^0 is

$$U_m^0 = U_m - \sum_{i=1}^{k-1} U_m^i = U_m \left(1 - \sum_{i=1}^{k-1} \frac{U_m^i}{U_m}\right) = U_m \left(1 - N \sum_{i=1}^{k-1} U_m^i\right).$$

With $\beta_m \coloneqq N \sum_{i=1}^{k-1} U_m^i$, this simplifies to

$$U_m^0 = U_m (1 - \beta_m) = \frac{1}{N} (1 - \beta_m).$$

We also compute the denominator of (35) as

$$\|U^0\|_1 = \sum_{m=0}^{N-1} |U_m^0| = \sum_{m=0}^{N-1} \frac{1}{N}(1 - \beta_m) = 1 - \sum_{m=0}^{N-1} \frac{1}{N} \beta_m.$$

Putting this into (35), we obtain

$$\Pi^*_m = U_m \frac{1-\beta_m}{1-\sum_{m=0}^{N-1}\frac{1}{N}\beta_m}. \tag{36}$$

In the remainder of the proof we aim at bounding $|\beta_m|$ from above by some $\beta = o(1)$. Then (36) gives

$$\frac{1-\beta_m}{1-\sum_{m=0}^{N-1}\frac{1}{N}\beta_m} \in \left[\frac{1-\beta}{1+\beta}, \frac{1+\beta}{1-\beta}\right] \subset \left[1-\frac{2\beta}{1-\beta}, 1+\frac{2\beta}{1-\beta}\right],$$

hence defining $\gamma = \frac{2\beta}{1-\beta} = o(1)$ proves the lemma.

To find the desired β with $|\beta_m| < \beta = o(1)$, we regard the vector $U - UQ$. On the one hand, its elements are very small. For all $m \in [0..N-1]$, we have

$$(U-UQ)_m = U_m - \sum_{\ell=0}^{N-1} q_\ell^m U_\ell = \frac{1}{N}\left(1 - \sum_{\ell=0}^{N-1} q_m^\ell\right) = \frac{q_m^N}{N},$$

where q_ℓ^m is the probability to go from individual ℓ to individual m (recall that $q_\ell^m = q_m^\ell$) and q_m^N is the probability to leave the plateau from the m-th individual. The Euclidean norm of this vector is also very small.

$$\begin{aligned}\|U-UQ\|_2 &= \sqrt{\sum_{m=0}^{N-1}\left(\frac{q_m^N}{N}\right)^2} \\ &= \frac{1}{N}\sqrt{\sum_{i=0}^{k-1}\binom{n}{k-i}\left(\Pr[\alpha = k-i]\Big/\binom{n}{k-i}\right)^2}.\end{aligned}$$

To bound the sum under the root we notice that it is maximized when we maximize $\Pr[\alpha = 1]$ (since we consider only constant k, we assume that n is large enough so that $n > 2k$). Let this probability be equal to 1, then we have only one non-zero summand and hence,

$$\|U-UQ\|_2 \le \frac{1}{N}\sqrt{\binom{n}{1}^{-1}} = \frac{1}{\sqrt{n}N}.$$

On the other hand, if we recall that the U^i are eigenvectors, then we have

$$U - UQ = \sum_{i=0}^{k-1} U^i - \sum_{i=0}^{k-1}\lambda_i U^i = \sum_{i=0}^{k-1}(1-\lambda_i)U^i.$$

As the U^i are orthogonal, for every $i \in [0..k-1]$ we have

$$\left\|(1-\lambda_i)U^i\right\|_2 \le \|U-UQ\|_2 \le \frac{1}{\sqrt{n}N}.$$

Since the absolute value of every component of a vector cannot be larger than its Euclidean norm, for all $m \in [0..N-1]$ we conclude that

$$|(1-\lambda_i)U^i_m| \leq \left\|(1-\lambda_i)U^i\right\|_2 \leq \frac{1}{\sqrt{n}N}.$$

Recall that by Lemma 66 we have

$$(1-\lambda_i) > \varepsilon > \frac{(\Pr[\alpha=1])^{k-1}}{(k-1)2^k}$$

for all $i \neq 0$. Consequently,

$$\begin{aligned}|\beta_m| &= \left|N\sum_{i=1}^{k-1} U^i_m\right| \leq N\sum_{i=1}^{k-1}|U^i_m| \leq N\sum_{i=1}^{k-1}\frac{1}{\sqrt{n}N(1-\lambda_i)}\\ &\leq \frac{(k-1)}{\sqrt{n}\varepsilon} =: \beta.\end{aligned}$$

Since we have $\Pr[\alpha=1] = \omega(n^{-\frac{1}{2k-2}})$, we conclude that

$$\varepsilon = \frac{\left(\omega(n^{-\frac{1}{2k-2}})\right)^{k-1}}{(k-1)2^k} = \omega(1/\sqrt{n}).$$

Thus,

$$\beta = \frac{(k-1)}{\sqrt{n}\varepsilon} = o(1)$$

as desired.

□

We are now in the position to prove our main result.

Proof of Theorem 21. To prove the theorem we first estimate the probability that the runtime T is greater than t by $\Pr[T > t] = \|\pi P^t\|_1$, where π is the initial distribution over the levels of the plateau. Then by (2) we estimate the expected runtime as $E[T] = \sum_{t=1}^{+\infty}\left\|\pi P^{t-1}\right\|_1$.

To analyse $\|\pi P^t\|_1$, we decompose π into a sum of eigenvectors of P using the basis $e^0,\ldots,e^{k-1}$ from Lemma 67. Let $\pi^0,\ldots,\pi^{k-1}$ be scalar multiples of $e^0,\ldots,e^{k-1}$ such that

$$\pi = \sum_{i=0}^{k-1}\pi^i. \tag{37}$$

Using the triangle inequalities (Lemma 54) we obtain

$$\left\|\pi^0P^t\right\|_1 - \left\|\sum_{i=1}^{k-1}\pi^iP^t\right\|_1 \leq \|\pi P^t\|_1 \leq \left\|\pi^0P^t\right\|_1 + \left\|\sum_{i=1}^{k-1}\pi^iP^t\right\|_1.$$

Since the π^i are the eigenvectors of P, we have

$$\lambda_0^t \left\|\pi^0\right\|_1 - \left\|\sum_{i=1}^{k-1} \lambda_i^t \pi^i\right\|_1 \leq \left\|\pi P^t\right\|_1 \leq \lambda_0^t \left\|\pi^0\right\|_1 + \left\|\sum_{i=1}^{k-1} \lambda_i^t \pi^i\right\|_1. \tag{38}$$

Now we estimate the "error term" $\left\|\sum_{i=1}^{k-1} \lambda_i^t \pi^i\right\|_1$ of these bounds. First, by Lemma 65 and by Lemma 62, we have

$$\left\|\sum_{i=1}^{k-1} \lambda_i^t \pi^i\right\|_1 = \left\|\phi\left(\sum_{i=1}^{k-1} \lambda_i^t \pi^i\right)\right\|_1 = \left\|\sum_{i=1}^{k-1} \lambda_i^t \phi(\pi^i)\right\|_1. \tag{39}$$

Using Lemma 53 and Lemma 55 we estimate

$$\left\|\sum_{i=1}^{k-1} \lambda_i^t \phi(\pi^i)\right\|_1 \leq \sqrt{N}\left\|\sum_{i=1}^{k-1} \lambda_i^t \phi(\pi^i)\right\|_2 \leq \sqrt{N} \max_{i\in[1..k-1]}(|\lambda_i^t|)\left\|\sum_{i=1}^{k-1} \phi(\pi^i)\right\|_2. \tag{40}$$

By Lemma 66 we have $\max_{i\in[1..k-1]}(|\lambda_i^t|) \leq (1-\varepsilon)^t$, where $\varepsilon = \frac{(\Pr[\alpha=1]))^{k-1}}{(k-1)2^k}$. Hence, by Lemma 56 and Lemma 53, we conclude

$$\sqrt{N} \max_{i\in[1..k-1]}(|\lambda_i^t|)\left\|\sum_{i=1}^{k-1} \phi(\pi^i)\right\|_2 \leq \sqrt{N}(1-\varepsilon)^t \left\|\phi(\pi)\right\|_1 = \sqrt{N}(1-\varepsilon)^t. \tag{41}$$

Finally, by (38), (39), (40), and (41) we obtain that

$$\left\|\pi P^t\right\|_1 = \lambda_0^t \left\|\pi^0\right\|_1 + r(t), \tag{42}$$

where $|r(t)| \leq \sqrt{N}(1-\varepsilon)^t$.

To estimate the expected runtime we put (42) into (2) and obtain

$$\begin{aligned} E[T] &= \sum_{t=1}^{+\infty} \left\|\pi P^{t-1}\right\|_1 = \sum_{t=1}^{+\infty} \left(\lambda_0^{t-1} \left\|\pi^0\right\|_1 + r(t-1)\right) \\ &= \sum_{t=1}^{+\infty} \lambda_0^{t-1} \left\|\pi^0\right\|_1 + \sum_{t=1}^{+\infty} r(t-1) = \frac{\left\|\pi^0\right\|_1}{1-\lambda_0} + \sum_{t=0}^{+\infty} r(t). \end{aligned} \tag{43}$$

By (41) we have

$$\left|\sum_{t=0}^{+\infty} r(t)\right| \leq \sum_{t=0}^{+\infty} |r(t)| \leq \sum_{t=1}^{+\infty} \sqrt{N}(1-\varepsilon)^t = \frac{(1-\varepsilon)\sqrt{N}}{\varepsilon} \leq \frac{\sqrt{N}}{\varepsilon}.$$

From the assumptions of the theorem we have

$$\varepsilon = \frac{(\Pr[\alpha=1])^{k-1}}{(k-1)2^k} = \frac{\omega(1/\sqrt{n})}{(k-1)2^k} = \omega(1/\sqrt{n}),$$

and hence

$$\left|\sum_{t=0}^{+\infty} r(t)\right| = o(\sqrt{nN}).$$

It remains to estimate $\left\|\pi^0\right\|_1$ and λ_0. Recall that by Lemma 65 we have $\left\|\pi^0\right\|_1 = \left\|\phi(\pi^0)\right\|_1$. From the linearity of ϕ (see Lemma 62) and (37) we obtain $\phi(\pi) = \sum_{i=0}^{k-1} \phi(\pi^i)$. Since $\pi^0, \dots \pi^{k-1}$ are scalar multiples of $e^0, \dots e^{k-1}$ and all $\phi(e^i)$ are orthogonal, we have a decomposition of $\phi(\pi)$ into a sum of orthogonal vectors. Therefore, by Lemma 57, we have

$$\left\|\phi(\pi^0)\right\|_1 = \frac{\langle \phi(\pi), \phi(e^0)\rangle}{\langle \phi(e^0), \phi(e^0)\rangle}. \tag{44}$$

By Lemma 68, the components of e^0 are almost equal to the components of the vector u of the uniform distribution in the level space[9]. Transferring this result to the individual space, we have $(\phi(e^0))_m = \frac{1}{N}(1+\gamma_j)$ for all $j \in [0..k-1]$ and all individuals m that belong to level j. This and the fact that by Lemma 65 we have $\|\phi(\pi)\|_1 = \|\pi\|_1 = 1$ allow to calculate the inner products in (44) and obtain

$$\left\|\pi^0\right\|_1 = \frac{\sum_{j=0}^{k-1} \sum_{m\in\text{level } j} (\phi(\pi))_m \frac{1+\gamma_j}{N}}{\sum_{j=0}^{k-1} \sum_{m\in\text{level } j} \left(\frac{1+\gamma_j}{N}\right)^2} = 1 + o(1). \tag{45}$$

We compute $(1-\lambda_0)$ in the following way. First, since $\|\pi^*\|_1 = 1$ and π^* is an eigenvector of P, we have

$$1 - \lambda_0 = 1 - \|\lambda_0 \pi^*\|_1 = 1 - \|\pi^* P\|_1 = 1 - \sum_{i=0}^{k-1} \left|\sum_{j=0}^{k-1} \pi_j^* p_j^i\right|.$$

Since by Theorem 20 all components of π^* are positive and the components of P are non-negative, this simplifies to

$$1 - \sum_{i=0}^{k-1} \left|\sum_{j=0}^{k-1} \pi_j^* p_j^i\right| = 1 - \sum_{i=0}^{k-1}\sum_{j=0}^{k-1} \pi_j^* p_j^i = 1 - \sum_{j=0}^{k-1} \pi_j^* \sum_{i=0}^{k-1} p_j^i.$$

By the definition of P, the sum of the j-th row of P is equal to $(1-p_j^k)$, and we have $\sum_{i=0}^{k-1} \pi_i^* = \|\pi^*\|_1 = 1$. Hence,

$$1 - \sum_{j=0}^{k-1} \pi_j^* \sum_{i=0}^{k-1} p_j^i = 1 - \sum_{j=0}^{k-1} \pi_j^* (1 - p_j^k) = \sum_{j=0}^{k-1} \pi_j^* p_j^k.$$

[9]Note that e^0 is the same vector as π^* in Lemma 68. However we now refer to this vector as e^0 to underline that we consider it as a basis vector of the level space, while in Lemma 68 we referred to it as π^* since we considered it as a vector of the probabilistic distribution over the states of the level chain.

By Lemma 68 and (29) we have

$$\sum_{j=0}^{k-1} \pi^*_j p_j^k = \sum_{j=0}^{k-1} \binom{n}{k-j} N^{-1}(1+\gamma_j) \binom{n}{k-j}^{-1} \Pr[\alpha = k-j]$$
$$= \frac{1}{N} \sum_{j=1}^{k} \Pr[\alpha = j](1+\gamma_{k-j}) = \frac{1}{N} \Pr[1 \le \alpha \le k](1+o(1)).$$

Thus, we obtain

$$\lambda_0 = 1 - \frac{1}{N} \Pr[1 \le \alpha \le k](1+o(1)). \tag{46}$$

By substituting λ_0 and $\left\|\pi^0\right\|_1$ into (43) and (42) with their values from (46) and (45) and recalling that $N = \frac{n^k}{k!}(1+o(1))$, we prove the theorem.

□

We also underline that $r(t)$ in the tail bounds on the runtime distribution is negligible, as soon as $t = \omega(\sqrt{n}\log(n))$, that is, far before the algorithm finds the optimum.

3.4 Corollaries

We now exploit Theorem 21 to analyze how the choice of the mutation operator influences the runtime.

By the runtime in this section we mean the runtime of the algorithm when it starts from an arbitrary individual, which is not necessarily on the plateau. However, without proof we notice that the $(1+1)$ EA with any mutation operator considered in this section reaches the plateau in an expected number of $O(n\log(n))$ iterations from any starting individual, which is significantly less than the time which it spends on the plateau. Therefore, the time to leave the plateau coincides with the total runtime precisely apart from lower order terms. Since, by our main result, the time to leave the plateau depends only on the probability to flip between 1 and k bits, determining the runtimes in this section is an easy task.

We first observe that for all unbiased operators with constant probability to flip exactly one bit, the expected optimization time is $\Theta(N)$, where we recall that the size N of the plateau is

$$N = \sum_{i=0}^{k-1} \binom{n}{n-k+i} = (1 \pm o(1)) \frac{n^k}{k!}.$$

Hence all these mutation operators lead to asymptotically the same runtime of $\Theta(n^k)$. The interesting aspect thus is how the leading constant changes.

3.4.1 Randomized Local Search and Variants

When taking such a more precise look at the runtime, that is, including the leading constant, then the best runtime, obviously, is obtained from mutation operators which flip always

between 1 and k bits. This includes variants of randomized local search which also flip more than one bit, see, e.g., [73, 111, 39], as long as they do not flip more than k bits, but most prominently the classic randomized local search heuristic, which always flips a single random bit. Note that the latter uniformly for all k (and including the case $k = 1$ not regarded in this chapter) is among the most effective algorithms.

3.4.2 Standard $(1+1)$ EA

The classic mutation operator in evolutionary computation is *standard bit mutation*, where each bit is flipped independently with some probability ("mutation rate") γ/n, where γ usually is a constant. We call the $(1+1)$ EA which uses the standard bit mutation the *standard* $(1+1)$ *EA*.

Theorem 22. *Let γ be some arbitrary positive constant and $k \geq 2$. Then the standard $(1+1)$ EA with mutation rate γ/n optimizes $\textsc{Plateau}_k$ in an expected number of*

$$E[T] = (1 + o(1)) \frac{n^k}{k! e^{-\gamma} \sum_{i=1}^{k} \frac{\gamma^i}{i!}}$$

iterations. This time is asymptotsically minimal for $\gamma = \sqrt[k]{k!} \approx k/e$.

Proof. For the standard bit mutation with mutation rate γ/n, the probability to flip exactly one bit is

$$n \frac{\gamma}{n} \left(1 - \frac{\gamma}{n}\right)^{n-1} \geq \gamma e^{-\gamma}(1 - o(1)),$$

which is at least some positive constant as long as γ is a constant. Thus, we can apply Theorem 21 and obtain

$$\begin{aligned} E[T] &= (1 \pm o(1)) \frac{N}{\Pr[1 \leq \alpha \leq k]} \\ &= (1 \pm o(1)) N \left(\sum_{i=1}^{k} \binom{n}{i} \left(\frac{\gamma}{n}\right)^i \left(1 - \frac{\gamma}{n}\right)^{n-i} \right)^{-1} \\ &= (1 \pm o(1)) \frac{n^k}{k!} \left(\sum_{i=1}^{k} \frac{\gamma^i}{i!} e^{-\gamma} \right)^{-1}. \end{aligned}$$

Consider $d(\gamma) = e^{-\gamma} \sum_{i=1}^{k} \frac{\gamma^i}{i!}$. In order to minimize $E[T]$, we have to maximize $d(\gamma)$. Now $\gamma \mapsto d(\gamma)$ is a smooth continuous function, so its maximal value for $\gamma \in [0, +\infty)$ can only be at $\gamma = 0$, for $\gamma \to +\infty$, or in the zeros of its derivative. We have $d(0) = \lim_{\gamma \to \infty} d(\gamma) = 0$. The derivative is

$$d'(\gamma) = \left(e^{-\gamma} \sum_{i=1}^{k} \frac{\gamma^i}{i!} \right)' = e^{-\gamma} \sum_{i=1}^{k} \frac{i\gamma^{i-1}}{i!} - e^{-\gamma} \sum_{i=1}^{k} \frac{\gamma^i}{i!}$$

$$= e^{-\gamma}\left(\sum_{i=0}^{k-1}\frac{\gamma^i}{i!} - \sum_{i=1}^{k}\frac{\gamma^i}{i!}\right) = e^{-\gamma}\left(1 - \frac{\gamma^k}{k!}\right).$$

Hence the only value of γ with $d'(\gamma) = 0$ is $\gamma = \sqrt[k]{k!}$. For this value we have $d(\sqrt[k]{k!}) > 0$, so this defines the unique optimal mutation rate. Finally, by Stirling's formula $k! \approx \sqrt{2\pi k}\left(\frac{k}{e}\right)^k$ we have $\sqrt[k]{k!} \approx (2\pi k)^{\frac{1}{2k}}\frac{k}{e} \approx \frac{k}{e}$. □

3.4.3 Fast $(1+1)$ EA

The *fast* $(1+1)$ *EA* proposed in [57] is simply a $(1+1)$ EA that uses standard bit mutation with a random mutation rate γ/n with $\gamma \in [1..n/2]$ chosen according to a power-law distribution. More precisely, for a parameter $\beta > 1$ which is assumed to be a constant (independent of n), we have

$$\Pr[\gamma = i] = 0$$

for every $i > n/2$ and $i = 0$, and

$$\Pr[\gamma = i] = i^{-\beta}/H_{n/2,\beta}$$

otherwise, where $H_{n/2,\beta} := \sum_{i=1}^{n/2} i^{-\beta}$ is a generalized harmonic number.

Theorem 23. *For $k \geq 2$ the expected runtime of the fast $(1+1)$ EA on $\textsc{Plateau}_k$ is $C_{kn}\frac{n^k}{k!}$, where $C_{kn} := \frac{H_{n/2,\beta}}{H_{k,\beta}}(1+o(1))$ can be bounded by constants, namely $C_{kn} \in \left[\frac{\frac{1}{\beta-1}-o(1)}{H_{k,\beta}}, \frac{\frac{1}{\beta-1}+1}{H_{k,\beta}}\right]$.*

Proof. From the definition of the fast $(1+1)$ EA we have

$$\sum_{i=1}^{k}\Pr[\gamma = i] = \frac{\sum_{i=1}^{k} i^{-\beta}}{\sum_{i=1}^{n/2} i^{-\beta}} = \frac{H_{k,\beta}}{H_{n/2,\beta}}.$$

Since $\beta > 1$ and k are constants, $H_{k,\beta}$ is a constant as well. We estimate $H_{n/2,\beta}$ through the corresponding integral.

$$\frac{1}{\beta-1} + 1 = \int_1^{+\infty} x^{-\beta}dx + 1 \geq H_{n/2,\beta} \geq \int_1^{n/2} x^{-\beta}dx = \frac{1-(n/2)^{1-\beta}}{\beta-1} = \frac{1-o(1)}{\beta-1}.$$

Notice that $\Pr[\alpha = 1] = \frac{H_{1,\beta}}{H_{n/2,\beta}} = \left(H_{n/2,\beta}\right)^{-1}$ is at least some constant. Thus, Theorem 21 gives an expected runtime of $E[T] = \frac{H_{n/2,\beta}}{H_{k,\beta}}N(1+o(1))$, which we can estimate by

$$\frac{\frac{1-o(1)}{\beta-1}}{H_{k,\beta}}\frac{n^k}{k!} \leq E[T] \leq \frac{\frac{1}{\beta-1}+1}{H_{k,\beta}}\frac{n^k}{k!}(1+o(1)).$$

□

3.4.4 Hyper-Heuristics

Hyper-heuristics are randomized search heuristics that combine, in a suitable and again usually randomized fashion, simple low-level heuristics. Despite many success stories in applications, their theoretical understanding is still very low and only the last few years have seen some first results. These exclusively regard simple $(1+1)$ type hill-climbers which choose between different mutation operators as low-level heuristics. We now regard the hyper-heuristics discussed in [1] argue that for some of these, our method is applicable, whereas for others it is not clear how to do this.

Like almost all previous theoretical works, we regard as available low-level mutation operators one-bit flips (flipping a bit chosen uniformly at random) and two-bit flips (flipping two bits chosen uniformly at random from all 2-sets of bit positions). Hence the $(1+1)$ hill-climber with this a hyper-heuristic selection between these two operators starts with a random search point and then repeats generating a new search point by applying one of the mutation operators (chosen according to the hyper-heuristic) and accepting the new search point if it has an at least as good fitness as the parent.

The most elementary hyper-heuristic called *simple random* in each iteration simply chooses one of the two available mutation operators with equal probability $1/2$. This compound mutation operator (choosing one randomly and applying it) still is a unary unbiased mutation operator, so our main result (Theorem 21) is readily applicable and gives the following result.

Theorem 24. *Consider the* $(1+1)$ *hill-climber using the* simple random *hyper-heuristic to decide between the one-bit flip and the two-bit flip mutation operator. When started on an arbitrary point of the plateau, its runtime T on the* $\textsc{Plateau}_k$*,* $k \geq 2$*, function satisfies*

$$E[T] = \frac{n^k}{k!}(1 \pm o(1)).$$

The more interesting hyper-heuristic *random gradient* in the first iteration chooses a random low-level heuristic. In each further iteration, it chooses the same low-level heuristic as in the previous iteration, if this has ended with a fitness gain, and it chooses again a random low-level heuristic otherwise. This way of performing mutation obviously cannot be described via a single unary unbiased operator. However, once the algorithm has reached the plateau, it can. The reason is that from that point on and until the optimum is found, no further improvements are found. Consequently, the algorithm reverts to the one using the *simple random* approach.

Theorem 25. *Consider the* $(1+1)$ *hill-climber using the* random gradient *hyper-heuristic to decide between the one-bit flip and the two-bit flip mutation operator. When started on an arbitrary point of the plateau, its runtime T on the* $\textsc{Plateau}_k$*,* $k \geq 2$*, function satisfies*

$$E[T] = \frac{n^k}{k!}(1 + o(1)).$$

For two other common hyper-heuristics, we currently do not see how to apply our methods. The *permutation* heuristic initially fixes a permutation of the low-level heuristics and then repeatedly uses them in this order. The *greedy* heuristic uses, in each iteration, all available hyper-heuristics in parallel and proceeds with the best offspring produced (if it is at least as

good as the parent). While we are optimistic that these heuristics lead to asymptotically the same runtimes as the two heuristics just analyzed, we cannot prove this since our main result is not applicable.

It has been observed in [102] that, due to the generally low probability of finding an improvement, better results are obtained when the *random gradient* heuristic is used with a longer learning period, that is, the randomly chosen low-level heuristic is repeated for a phase of τ iterations. If an improvement is found, a new phase with the same low-level heuristic is started. Otherwise, the next phase starts with a random operator. This idea was extended in [60] so that now a phase was called successful if within τ iterations a certain number σ of improvements were obtained. This mechanism was more stable and allowed a self-adjusting choice of the previously delicate parameter τ. Again, for these hyper-heuristics our results are not applicable.

3.4.5 Comparison for Concrete Values

Since the leading constants computed above, in their general form, are hard to compare, we now provide in Table 1 a few explicit values for specific algorithm parameters and plateau sizes.

Algorithm		$k=2$	$k=4$	$k=6$
Random Local Search		1	1	1
$(1+1)$ EA with standard bit mutation	Mutation rate $\frac{1}{n}$	1.812	1.591	1.582
	Mutation rate $\frac{k}{en}$	2.074	1.328	1.027
Fast Genetic Algorithm	$\beta = 1.5$	1.930	1.563	1.428
	$\beta = 2$	1.316	1.155	1.103
$(1+1)$ EA with hyperheuristics	simple random	1	1	1
	random gradient	1	1	1

Table 1 – *Comparison of the leading constant in the expected runtime of the evolutionary algorithms with different mutation operators on the* $\textsc{Plateau}_k$ *function, that is, the constant c, such that the expected runtime is* $c\frac{n^k}{k!}(1-o(1))$.

3.5 Parallel Runs and Population-Based Algorithms

In this section we show how our precise tail bound for the runtime can be useful when it comes to the analysis of the algorithms other than $(1+1)$ EA. First, we consider λ parallel runs of the $(1+1)$ EA with $\lambda = \Theta(1)$. We call the runtime of these parallel runs $\tilde{T}$ and define it as the minimal $t \in \mathbb{N}$ such that at least one of λ processes finds the optimum in t iterations. In other

words, denoting by T_i the runtime of the i-th process, we have

$$\tilde{T} = \min_{i \in [1..\lambda]} T_i.$$

We show in the following theorem that $\tilde{T}$ follows a distribution close to $\frac{1}{\lambda}\,\mathrm{Geom}(p)$ with $p = \frac{\Pr[1 \le \alpha \le k] k!}{n^k}(1 + o(1))$.

Theorem 26. *Let $\tilde{T}$ be the parallel runtime until the optimum of the $\textsc{Plateau}_k$ function is found by one of $\lambda = \Theta(1)$ parallel runs of the $(1+1)$ EA with an unbiased mutation operator such that $\Pr[1 \le \alpha \le k] = \omega(n^{-\frac{1}{2k-2}})$. Then we have*

$$\Pr[\tilde{T} > t] = (1-p)^{\lambda t}(1 + s(t)),$$

where $p = \frac{\Pr[1 \le \alpha \le k] k!}{n^k}(1 + o(1))$ and $s(t) = o(1)$ for all $t \ge \frac{2k \ln(n)}{\varepsilon}$, where ε is the same as in Theorem 21.

To simplify the proof of the theorem we first prove the following auxiliary lemma.

Lemma 69. *With ε defined in Theorem 21 and p defined in Theorem 26 we have*

$$\left(\frac{1-\varepsilon}{1-p}\right)^{\frac{2}{\varepsilon}} \le e^{-1}.$$

Proof. First notice that since $\varepsilon \in [0,1]$, we have $e^{\varepsilon/2} \le (1+\varepsilon)$. Hence, we have

$$(1-\varepsilon)e^{\frac{\varepsilon}{2}} \le (1-\varepsilon)(1+\varepsilon) \le 1 - \varepsilon^2 \le 1 - p,$$

if n is large enough, since $\varepsilon = \omega(n^{-1/2})$ and $p = O(n^{-k})$. By multiplying both sides of the inequality by $\frac{1}{e^{\varepsilon/2}(1-p)}$ and then raising them in power of $\frac{2}{\varepsilon}$, we prove the lemma. □

We are in a position to prove Theorem 26.

Proof of Theorem 26. To have $\tilde{T} > t$ we need all λ processes not to find the optimum in t iterations. Recall that T_i is the runtime of the i-th process. Since the processes are independent and they all have the same distribution of the runtime, we have

$$\Pr[\tilde{T} > t] = \prod_{i=1}^{\lambda} \Pr[T_i > t] = (\Pr[T > t])^{\lambda},$$

where T is the runtime of the $(1+1)$ EA with the same mutation operator. Notice that T is a subject for Theorem 21 which gives us

$$\Pr[\tilde{T} > t] = ((1 + o(1))(1-p)^t + r(t))^{\lambda} = (1-p)^{\lambda t}\left(1 + \frac{r(t)}{(1-p)^t} + o(1)\right)^{\lambda}.$$

If $t \ge \frac{2k\ln(n)}{\varepsilon}$, then by Lemma 69 we have

$$\left|\frac{r(t)}{(1-p)^t}\right| \le \frac{\sqrt{N}(1-\varepsilon)^t}{(1-p)^t} = \sqrt{N}\left(\frac{1-\varepsilon}{1-p}\right)^{\frac{2k\ln(n)}{\varepsilon}}$$

$$\leq \sqrt{Ne}^{-k\ln(n)} = O(n^{k/2})n^{-k} = o(1).$$

Therefore, since we only consider $\lambda = \Theta(1)$, for these values of t we have

$$\Pr[\tilde{T} > t] = (1-p)^{\lambda t}(1+o(1))^{\lambda} = (1-p)^{\lambda t}(1+o(1)).$$

□

Unfortunately, we cannot state the same result for the super-constant λ. The root of this problem is that when we have a super-constant number of points on plateau, the approximation of the uniform distribution with any other distribution given in Lemma 68 is not enough. Namely, the factor of $\left\|\pi^0\right\|_1$ can be no longer $(1+o(1))$, when it is raised in a super-constant power of λ.

Nevertheless, we still can analyse the runtime of the $(\lambda \overset{1:1}{+} \lambda)$ EA with constant population size. For this we consider the $(\lambda \overset{1:1}{+} \lambda)$ EA with an additional diversity mechanism, which affects how we break ties when we select individuals to the next population. With this mechanism, in a case of a tie we select individuals into the next population in the following order.

1) First, we choose individuals from the offspring population.
2) Second, we choose parents whose offspring was not selected into the next population.
3) Finally, we choose other individuals from the parent population.

The idea of maintaining diversity with this mechanism comes from the observation that an offspring is usually not significantly different from its parent. With these tie-breaking rules we select a pair of a parent and its offspring into the next population only if there exists another pair with smaller fitness. This prevents us from selecting two individuals with similar genotype into the next population, making us prefer individuals with different genotype.

One can see that as soon as all individuals of the $(\lambda \overset{1:1}{+} \lambda)$ EA reach the plateau, it turns into λ independent runs of the $(1+1)$ EA. This happens because when all individuals are already in the plateau, then in each pair of an offspring and its parent there is at least one individual with fitness at least $n-k$, hence, exactly one of them satisfies one of the first two rules. This leads us to the following theorem.

Theorem 27. *Let $\lambda = \Theta(1)$ and let T be the runtime of the $(\lambda \overset{1:1}{+} \lambda)$ EA with the described diversity mechanism on $\textsc{Plateau}_k$ if it starts with the whole population on the plateau. If the $(\lambda \overset{1:1}{+} \lambda)$ EA uses an unbiased mutation operator such that $\Pr[1 \leq \alpha \leq k] = \omega(n^{-\frac{1}{2k-2}})$, then we have*

$$\Pr[T > t] = (1-p)^{\lambda t}(1+s(t)),$$

where $p = \frac{\Pr[1\leq\alpha\leq k]k!}{n^k}(1+o(1))$ and $s(t) = o(1)$ for all $t \geq \frac{2k\ln(n)}{\varepsilon}$, where ε is the same as in Theorem 21.

Finally, we note that the $(\lambda \overset{1:1}{+} \lambda)$ EA with $\lambda = \Theta(1)$ reaches the plateau with at least one individual in $O(n^{1+\frac{1}{2k-2}}\log(n))$ iterations (see Subsection 2.2.4). Then the individual in the plateau creates an offspring lying in the plateau with probability $\Omega(n^{-1-\frac{1}{2k-2}})$, and therefore, all population reaches the plateau in at most $\lambda\Omega(n^{1+\frac{1}{2k-2}}) = \Omega(n^{1+\frac{1}{2k-2}})$ iterations. By Theorem 27 this time is much smaller than the time the algorithm spends on the plateau.

3.6 Conclusion of Chapter 3

In this chapter we developed a new method to analyze the runtime of evolutionary algorithms on plateaus. This method does not depend on the particular mutation operator used by the EA as long as there is a sufficiently large probability to flip a single random bit. We performed a very precise analysis on the particular class of plateau functions, but we are optimistic that similar methods can be applied for the analysis of other plateaus. For example, Lemmas 66, 67 and 68 remain true for those plateaus of the function XDIVK (that is defined as $\lfloor \textsc{OneMax}(x)/k \rfloor$ for some parameter k) that are in a constant Hamming distance from the optimum (and these are the plateaus which contribute most to the runtime). That said, the proof of Lemma 66 would need to be adapted to these plateaus different from the one of our plateau function. We are optimistic that this can be done, but leave it as an open problem for now.

The inspiration for our analysis method stems from the observation that the algorithm spends a relatively long time on the plateau. So regardless of the initial distribution on the plateau, the distribution of the individual converges to the conditional stationary distribution long before the algorithm leaves the plateau. This indicates that our method is less suitable to analyze how evolutionary algorithms leave plateaus which are easy to leave, but such plateaus usually present not bigger problems in optimization.

Overall, we are optimistic that our main analysis method, switching between the level chain and the individual chain, which might be the first attempt to devise a general analysis method for EAs on plateaus, will find further applications.

The precision of the obtained results allowed us to estimate the distribution of the population-based $(\lambda \overset{1:1}{+} \lambda)$ EA with a diversity mechanism. However, it is still an open question how the more complicated EAs like perform on plateaus. A series of works [81, 80, 87, 138, 84, 13, 121, 49, 33, 31] analyzing the runtime of various versions of the $(\mu + \lambda)$ EA, (μ, λ) EA, and $(1 + (\lambda, \lambda))$ GA on ONEMAX show that these algorithms quickly reach the plateau of the PLATEAU function, but it is currently not clear how to extend our method to get sharp runtime estimates also for the part of the process on the plateau. Likewise, it is not clear how our methods can be extended to algorithms that dynamically change their parameters [32], because here in most cases the relevant state of the algorithm not only consists of the current search point(s). For hyper-heuristics, we could show two elementary results, but again, as discussed in Section 3.4.4, for most hyper-heuristics our general result cannot be applied. By analogy with jump functions, crossover-based algorithms should be efficient on plateaus, especially the ones using different diversity mechanisms [20]. However, these algorithms are more complicated, and even on jump functions there are no asymptotically tight bounds on their runtime. This suggest that studying their behavior on plateaus might be even more complicated. Given that plateaus of constant fitness appear frequently in optimization problems, we feel that the open questions discussed in this paragraph are worth pursuing in the near future.

Chapter 4 The Methods for Analysis of the Crossover-Based Algorithms

In this chapter our focus is on the analysis of a relatively new algorithm—the $(1 + (\lambda, \lambda))$ GA proposed in [34]. This algorithm is of a special interest for us, since it not only uses non-trivial populations, but also the crossover operator. Being the first crossover-based algorithm which can solve OneMax in a linear time, this algorithm requires a non-trivial parameter control methods to reach its best efficiency. In [11] it was shown that the parameter control technique used to obtain a linear runtime on OneMax does not work well on the MAX-3SAT instances and needs additional adjustment for a reasonable runtime. On the other hand, in [2] it was empirically observed that such an adjusted version of the $(1 + (\lambda, \lambda))$ GA does not work well when the distance to the optimum is small (e.g., in case of re-optimization after a small change of a fitness function). This raises a question if there exist a more universal way of a dynamic parameter choice for the $(1 + (\lambda, \lambda))$ GA.

In this chapter we propose a modification of the $(1 + (\lambda, \lambda))$ GA which we call the *heavy-tailed* $(1 + (\lambda, \lambda))$ *GA*. This modified version simply chooses the parameters of the algorithm in each iteration from the power-law distribution. Our analysis of this algorithm on the OneMax, LeadingOnes and Jump_k functions shows that simple way of the dynamic parameters choice can be surprisingly effective. We also show that variation of the parameters of the power-law distribution does not have a strong impact on the runtime, hence it is fair to call the heavy-tailed $(1 + (\lambda, \lambda))$ GA a parameterless algorithm.

Previously the $(1 + (\lambda, \lambda))$ GA has not been analysed neither on LeadingOnes, nor on Jump_k, so we also derived the runtime of the standard $(1 + (\lambda, \lambda))$ GA with static parameters on these functions. From the analysis on the LeadingOnes we see that the $(1 + (\lambda, \lambda))$ GA struggles from the weak fitness-distance correlation of the optimized function, however, it does not become worse than the standard mutation-based algorithms. The study of the $(1 + (\lambda, \lambda))$ GA on Jump_k showed that the standard parameter setting of the algorithm can be not optimal when we optimize non-unimodal functions (in the same manner as it was shown in [57] that the standard recommendation to use mutation rate $\frac{1}{n}$ does not work well for Jump_k).

From the technical point of view our analysis is not too complicated (compared to the previous chapters), however to deliver the runtime on LeadingOnes we had to develop a new drift theorem which can deliver tail bounds under additive drift. This new theorem resembles some of the results in [92], being less general while easier for the application.

The structure of this chapter is as follows. First we describe the heavy-tailed $(1 + (\lambda, \lambda))$ GA in Section 4.1. Then we analyse it on OneMax function and provide its empirical comparison with its natural competitors on OneMax and on MAX-3SAT instances used in [11] in Section 4.2. We proceed with analysis of the $(1 + (\lambda, \lambda))$ GA with static parameters and of the heavy-tailed $(1 + (\lambda, \lambda))$ GA on LeadingOnes in Section 4.3 starting it with stating and proving our new drift theorem. Finally, in Section 4.4 we analyse the $(1 + (\lambda, \lambda))$ GA with static parameters on jump and then propose a slightly different version of the heavy-tailed $(1 + (\lambda, \lambda))$ GA for the non-unimodal functions in Section 4.5.

4.1 Heavy-tailed $(1 + (\lambda, \lambda))$ GA

It is often cited as a strength of evolutionary algorithms (EAs) that by setting the parameters right the algorithm can be adjusted to the particular problem to be solved. However, it is also known that this process of optimizing the parameters is time-consuming and needs a lot of expert knowledge.

The theoretical research in this field (see, e.g., [4, 51, 86, 112]) has contributed to this challenge via mathematical runtime analyses for general parameter values which allow to understand the influence of the parameter on the performance and to derive optimal parameter values. Examples (already discussed in the previous chapters) include (i) the works of Jansen, de Jong, and Wegener [87] as well as Doerr and Künnemann [49], which determine the runtime of the $(1 + \lambda)$ EA on ONEMAX for general value of λ and from this derive that a linear speed-up exists only for $\lambda = O(\frac{\log(n)\log\log(n)}{\log\log\log(n)})$, (ii) Witt's analysis [138] of the runtime of the $(\mu + 1)$ EA for general values of μ on the LEADINGONES benchmark, which in particular shows that for $\mu = O(\log n)$ a larger parent population does not lead to an asymptotic slow-down of the algorithm, or (iii) the results of Lehre [96, 95] and many follow-up works, which for many non-elitist algorithms determine asymptotically precise thresholds for the selection pressure that separate a highly inefficient regime from one with polynomial runtimes.

Concerning the mutation rate p of the standard bit mutation operator for bit strings of length n, which is one of our main objects of interest in this chapter, a large number of classic results suggests that a value of $p = \frac{1}{n}$ or close by is a good choice. We note that a mutation rate of $p = \frac{1}{n}$ means that on average a single bit is flipped. The recommendation $p = \frac{1}{n}$ can already be found in [5, 108]. Rigorously proven results show, among others, that only $p = \Theta(\frac{1}{n})$ can give an $O(n \log n)$ runtime of the $(1 + 1)$ EA on ONEMAX [67], that the asymptotically optimal mutation rate for the $(1 + 1)$ EA on LEADINGONES is approximately $p = \frac{1.59}{n}$, that $p = (1 \pm o(1))\frac{1}{n}$ is the asymptotically best mutation rate of the $(1 + 1)$ EA for all pseudo-Boolean linear functions [139], that only a mutation rate below $\frac{c}{n}$, where c is a specific constant, guarantees a polynomial runtime of the $(1 + 1)$ EA on all monotonic functions [59, 100], and that $(1 \pm o(1))\frac{1}{n}$ is the optimal mutation rate for the $(1 + \lambda)$ EA on ONEMAX when λ is small [75].

In the light of this previous state of the art, it came as a surprise when Doerr, Le, Makhmara, and Nguyen [57] determined the runtime of the $(1 + 1)$ EA on jump functions for general mutation rates and observed that here much higher mutation rates were optimal. It showed that for this multimodal benchmark function the insight gained on unimodal functions like ONEMAX, linear functions, or LEADINGONES do not apply. The optimal mutation rate for JUMP_{nk} was found to be $(1 \pm o(1))\frac{k}{n}$. The work [57] also showed that deviating from this optimal rate by a small constant factor already leads to a runtime increase by a factor of $e^{\Omega(k)}$. Consequently, the choice of the mutation rate for this problem is truly delicate.

To overcome this difficulty, the use of a random mutation rate chosen according to a heavy-tailed distribution, more specifically, a power-law distribution with exponent $\beta > 1$, was suggested. This mutation operator, called *fast mutation* in agreement with previous uses of heavy-tailed distributions in continuous evolutionary computation, samples a random number $\alpha \in [1..\lfloor \frac{n}{2} \rfloor]$ with probability proportional to $\alpha^{-\beta}$ and then flips each bit independently with rate $\frac{\alpha}{n}$. Each application of this operator independently samples a new value of α.

The main result in [57] is that the $(1 + 1)$ EA with this mutation operator for all k optimizes JUMP_{nk} in a time that is only by a factor of $O(k^{\beta-0.5})$ larger than the time resulting from

Algorithm 5 – The heavy-tailed $(1+(\lambda,\lambda))$ GA with power-law exponent β and upper limit u maximizing $f : \{0,1\}^n \to \mathbb{R}$

$x \leftarrow$ random bit string of length n
while not terminate **do**
    Choose λ from $[1..u]$ with $\Pr[\lambda = i] \sim i^{-\beta}$
Mutation phase:
    Choose $\ell \sim \mathrm{Bin}(n,p)$
    for $i \in [1..\lambda]$ **do**
        $x^{(i)} \leftarrow$ a copy of x
        Flip ℓ bits in $x^{(i)}$ chosen uniformly at random
    end for
    $x' \leftarrow \arg\max_{z \in \{x^{(1)},\dots,x^{(\lambda)}\}} f(z)$
Crossover phase:
    for $i \in [1..\lambda]$ **do**
        Create $y^{(i)}$ by taking each bit from x' with probability c and by taking it from x with probability $(1-c)$
    end for
    $y \leftarrow \arg\max_{z \in \{y^{(1)},\dots,y^{(\lambda)}\}} f(z)$
    if $f(y) \geq f(x)$ **then**
        $x \leftarrow y$
    end if
end while

standard bit mutation with the optimal rate. Given that missing the optimal rate (which is only accessible when knowing k) by a small constant factor already incurs a runtime increase by a factor of $e^{\Omega(k)}$, the $O(k^{\beta-0.5})$ price for having a one-size-fits-all mutation operator appears to be a good investment. From the asymptotic point of view β should be taken arbitrarily close to 1, but the experiments conducted in [57] suggested that $\beta = 1.5$ is a good choice. Both theory and experiments showed that the choice of β is not overly critical. For this reason, it is fair to call fast mutation a parameterless operator.

Since the fast mutation operator is nothing else than a random linear combination of standard bit mutation operators with rates $\frac{\alpha}{n}, \alpha = 1,\dots,\lfloor \frac{n}{2} \rfloor$, it is not surprising that the resulting runtime is higher than the one from the best of these individual operators. Rather, it is surprising that by simply averaging over the available options, one comes relatively close to the optimum, and this in a scenario where for static rates a small deviation from the optimum leads to a significantly increased runtime.

The reason why the fast mutation is so effective on JUMP is that it has a probability of at least $\Omega(k^{-(\beta-0.5)})$ to choose a mutation rate close to the optimal one, while we are not punished for missing the right rate. We find such behavior also profitable in a different situation. Namely, it was shown in [33] that when we optimize ONEMAX with the $(1+(\lambda,\lambda))$ GA, the optimal value of λ grows as we approach the optimum. However, if we miss the optimal λ, we are still likely to have some progress. Hence, it is natural to choose parameter λ from a power-law distribution. Hence, we come up with a modification of the $(1+(\lambda,\lambda))$ GA shown in Algorithm 5.

4.2 Runtime Analysis of the Heavy-Tailed $(1 + (\lambda, \lambda))$ GA on OneMax

The main result of this section is that not only the use of the heavy-tailed parameter choice in the $(1 + (\lambda, \lambda))$ GA relieves us from finding a good mutation rate, but surprisingly we can even obtain a runtime that is faster than the runtime of the $(1 + (\lambda, \lambda))$ GA with any fixed mutation rate: If the power-law exponent β satisfies $2 < \beta < 3$, then the heavy-tailed $(1 + (\lambda, \lambda))$ GA has an expected runtime of $O(n)$ on OneMax.

We note that a linear runtime of the $(1 + (\lambda, \lambda))$ GA on OneMax was obtained earlier with a self-adjusting choice of the mutation rate based on the one-fifth rule [31]. While this worked well on OneMax, experimental [76] and theoretical [11] studies on MAX-3SAT instances showed that this approach carries the risk that the population size λ increases rapidly because the problem structure may just not allow a one-fifth success rate, regardless how large λ is. Since this behavior increases the time complexity of each iteration, it leads to a significant performance loss. Such problems, naturally, cannot arise with the static behavior of the fast mutation operator.

Via an empirical study, we show that the fast mutation operator indeed without any modification also solves well the MAX-3SAT instances for which the one-fifth rule variant of the $(1 + (\lambda, \lambda))$ GA did not perform well in [11] (unless enriched with a suitable cap on λ). However, our study also shows that on OneMax itself, the self-adjusting $(1 + (\lambda, \lambda))$ GA is by a constant factor faster than the heavy-tailed $(1 + (\lambda, \lambda))$ GA. Since the runtime loss from a degenerate behavior of the one-fifth rule version of the $(1 + (\lambda, \lambda))$ GA can be large (due to the population size of order n), we draw from these results the recommendation to use the more robust heavy-tailed $(1 + (\lambda, \lambda))$ GA on a novel problem rather than the self-adjusting $(1 + (\lambda, \lambda))$ GA.

4.2.1 Runtime Analysis

In this section we prove upper and lower bounds on the runtime of the heavy-tailed $(1 + (\lambda, \lambda))$ GA on OneMax.

Upper Bound

Our goal in this subsection is to prove an upper bound on the number of fitness evaluations taken until the heavy-tailed $(1 + (\lambda, \lambda))$ GA finds the optimum of the OneMax benchmark. Since it is technically easier, we first regard the number of iterations until the optimum is found. For algorithms with fixed population sizes, such a bound would immediately imply a bound on the number of fitness evaluations (namely by multiplying the number of iterations with the fixed number of fitness evaluations per iteration). For the heavy-tailed $(1 + (\lambda, \lambda))$ GA using a newly sampled value of λ in each iteration, things are not that easy, but Wald's equation (Lemma 20) allows to argue that multiplying with the expected number of fitness evaluations per iteration gives the right result.

We start by showing that for reasonable parameter values, the optimum is found in a linear number of iterations.

Theorem 28. *If $\beta \in (1,3)$ and $u \geq \ln^{\frac{1}{3-\beta}}(n)$, then the expected number of iterations until the heavy-tailed $(1+(\lambda,\lambda))$ GA finds the optimum of OneMax function is $O(n)$.*

When $\beta > 2$, the expected number of fitness evaluations per iteration is constant (see Lemma 6). With this observation and Wald's equation, we obtain the following estimate for the runtime.

Theorem 29. *If $\beta \in (2,3)$ and $u \geq \ln^{\frac{1}{3-\beta}}(n)$, then the expected number of fitness evaluations until the heavy-tailed $(1+(\lambda,\lambda))$ GA finds the optimum of OneMax function is $O(n)$.*

We start with the proof of Theorem 28. For the readers' convenience we split the proof into Lemmas 70 and 71. The first lemma is essentially an interpretation of Lemma 7 in [33].

Lemma 70. *If $\lambda \leq \sqrt{\frac{n}{d(x)}}$, where $d(x)$ is the current distance between the current individual x and the optimum, then the probability $p_{d(x)}(\lambda)$ of increasing the fitness in one iteration is at least*

$$C\frac{d(x)\lambda^2}{n},$$

where $C > 0$ is an absolute constant. If $\lambda > \sqrt{\frac{n}{d(x)}}$, then this probability is at least C.

Proof. By [33, Lemma 7], the probability of a true progress $p_{d(x)}(\lambda)$ is at least

$$C'\left(1-\left(1-\frac{d(x)}{n}\right)^{\frac{\lambda^2}{2}}\right),$$

where $C' > 0$ is an absolute constant. By Lemma 21 we have

$$p_{d(x)}(\lambda) \geq C'\left(1-\left(1-\frac{d(x)}{n}\right)^{\frac{\lambda^2}{2}}\right) \geq C'\frac{\frac{d(x)\lambda^2}{2n}}{1+\frac{d(x)\lambda^2}{2n}}.$$

If $\lambda \leq \sqrt{\frac{n}{d(x)}}$, then we have $p_{d(x)}(\lambda) \geq C'\frac{d(x)\lambda^2}{3n}$. Note that $C := \frac{C'}{3}$ is an absolute constant as well as C'. If $\lambda > \sqrt{\frac{n}{d(x)}}$, then $p_{d(x)}(\lambda) \geq \frac{C'}{3} = C$.

Since Lemma 7 in [33] is formulated for $\lambda \geq 2$ only, we also note that for $\lambda = 1$ the algorithm essentially performs an iteration of the $(1+1)$ EA. Therefore, the probability for a progress in this case is at least $\frac{d(x)}{en}$. □

Lemma 71. *Let $\beta \in (1,3)$. Then the probability $p_{d(x)}$ of having progress in one iteration given that the current distance to the optimum is $d(x)$ is at least*

$$C(\beta)\frac{d(x)U^{3-\beta}}{n},$$

where $U = \min\{u, \sqrt{\frac{n}{d(x)}}\}$ and $C(\beta)$ is some constant independent of n.

β	$u \leq \sqrt{\frac{n}{d(x)}}$	$u > \sqrt{\frac{n}{d(x)}}$
< 1	$\Omega\left(\frac{d(x)u^2}{n}\right)$	$\Omega(1)$
$= 1$	$\Omega\left(\frac{d(x)u^2}{n\log(u)}\right)$	$\Omega\left(\frac{1}{\ln(u)} + \left(1 - \frac{\ln(n/d(x))}{2\ln(u)}\right)\right)$
$(1,3)$	$\Omega\left(\frac{d(x)u^{3-\beta}}{n}\right)$	$\Omega\left(\sqrt{\frac{n}{d(x)}}^{-1-\beta}\right)$
$= 3$	$\Omega\left(\frac{d(x)\log(u)}{n}\right)$	$\Omega\left(\frac{\log(n/d(x))}{(n/d(x))}\right)$
> 3	$\Omega\left(\frac{d(x)}{n}\right)$	

Table 2 – *The probability $p_{d(x)}$ to increase fitness in one iteration for various values of parameters $\beta \in \mathbb{R}$ and $u \in \mathbb{N}$*

Proof. By Lemma 70 we have

$$\begin{aligned} p_{d(x)} &= \sum_{\lambda=1}^{u} C_{\beta,u}\lambda^{-\beta} p_{d(x)}(\lambda) \geq C_{\beta,u} C \sum_{\lambda=1}^{\lfloor U \rfloor} \frac{d(x)\lambda^{2-\beta}}{n} \\ &= C_{\beta,u} C \frac{d(x)}{n} \sum_{\lambda=1}^{\lfloor U \rfloor} \lambda^{2-\beta} \end{aligned}$$

If $U \geq 2$, then by Lemma 3 we have

$$\sum_{\lambda=1}^{\lfloor U \rfloor} \lambda^{2-\beta} \geq \frac{U^{3-\beta}-1}{3-\beta} \geq \frac{1-2^{\beta-3}}{3-\beta} U^{3-\beta} \geq \frac{3}{8} U^{3-\beta}.$$

Otherwise, when $U < 2$ we have

$$\sum_{\lambda=1}^{\lfloor U \rfloor} \lambda^{2-\beta} \geq 1 = U^{\beta-3} U^{3-\beta} \geq 2^{\beta-3} U^{3-\beta} \geq \frac{1}{4} U^{3-\beta}.$$

Finally, we estimate

$$\begin{aligned} p_{d(x)} &\geq C_{\beta,u} C \frac{d(x)}{n} \sum_{\lambda=1}^{\lfloor U \rfloor} \lambda^{2-\beta} \\ &\geq C_{\beta,u} C \frac{1}{4} \frac{d(x)}{n} U^{3-\beta} = C(\beta) \frac{d(x) U^{3-\beta}}{n} \end{aligned}$$

with $C(\beta) \coloneqq \frac{1}{4} C_{\beta,u} C$.

□

In order to show a full picture we also computed the values of $p_{d(x)}$ for a wider range of parameters u and β. The results are shown in Table 2. We omit the proofs, since they are similar to the proof of Lemma 71.

We are now ready to prove Theorem 28.

Proof of Theorem 28. By Theorem 3 we estimate the upper bound on the expectation of the runtime T_I (in terms of iterations) as the sum of expected times until the algorithm leaves each fitness level. By Lemma 71 we have

$$\begin{aligned} E[T_I] &\leq \sum_{d(x)=1}^{n} \frac{1}{p_{d(x)}} \\ &\leq \frac{1}{C(\beta)} \left(\sum_{d(x)=1}^{\lfloor n/u^2 \rfloor} \frac{n}{d(x)u^{3-\beta}} + \sum_{d(x)=\lfloor n/u^2 \rfloor+1}^{n} \sqrt{\frac{n}{d(x)}}^{\beta-1} \right). \end{aligned}$$

By Lemma 4 we estimate the first sum

$$\sum_{d(x)=1}^{\lfloor n/u^2 \rfloor} \frac{n}{d(x)u^{3-\beta}} \leq \frac{n\left(\ln\left(\frac{n}{u^2}\right)+1\right)}{u^{3-\beta}} \leq \frac{n(\ln(n)+1)}{\ln(n)} = n(1+o(1)),$$

where in the last inequality we used the assumption $u \geq \ln^{\frac{1}{3-\beta}}(n)$. By Lemma 4 we estimate the second sum as follows.

$$\begin{aligned} \sum_{d(x)=\lfloor n/u^2 \rfloor+1}^{n} \sqrt{\frac{n}{d(x)}}^{\beta-1} &\leq \sum_{d(x)=1}^{n} \sqrt{\frac{n}{d(x)}}^{\beta-1} \\ &\leq n^{\frac{\beta-1}{2}} \sum_{d(x)=1}^{n} d(x)^{-\frac{\beta-1}{2}} \leq n^{\frac{\beta-1}{2}} \frac{n^{\frac{3-\beta}{2}}}{(3-\beta)/2} = O(n). \end{aligned}$$

Therefore, we have

$$E[T_I] \leq \frac{1}{C(\beta)} \left(O(n)+O(n)\right) = O(n). \qquad \square$$

We are now in position to prove Theorem 29

Proof of Theorem 29. Let $\{\lambda_t\}_{t\in\mathbb{N}}$ be a sequence of random variables, each following the power-law distribution with parameters β and u. We can assume that for all $t \in \mathbb{N}$ the heavy-tailed $(1+(\lambda,\lambda))$ GA chooses $\lambda := \lambda_t$ in iteration t. Since the cost of one iteration is 2λ fitness evaluations (λ for the mutation phase and λ for the crossover phase), the total number of fitness evaluations T_F has the same distribution as $\sum_{t=1}^{T_I} 2\lambda_t$. We aim at proving that the sequence $(\lambda_t)_{t\in\mathbb{N}}$ and T_I allow to use Wald's equation (Lemma 20). We show that conditions (1)–(4) of this lemma are satisfied.

1) All λ_t have the same expectation, which is finite by Lemma 6.
2) The event $T_I \geq t$ is independent of the outcome of λ_t, which implies that for all $i \in [1..u]$ we have $\Pr[T_I \geq t \mid \lambda_t = i] = \Pr[T_I \geq t]$. Therefore, we have

$$\begin{aligned} E[\lambda_t \mathbb{1}_{\{T_I \geq t\}}] &= \sum_{i=1}^{u} i \Pr[\lambda_t = i] \Pr[T_I \geq t \mid \lambda_t = i] \\ &= \Pr[T_I \geq t] \sum_{i=1}^{u} i \Pr[\lambda_t = i] = \Pr[T_I \geq t] E[\lambda_t]. \end{aligned}$$

β	$E[T_I]$	$E[T_F] = 2E[T_I]E[\lambda]$
< 1	$O(n)$ if $u \geq \sqrt{\ln(n)}$ $O\left(\frac{n}{u^2}\log\frac{n}{u^2}\right)$ if $u \leq \sqrt{\ln(n)}$	$O(nu^{2-\beta})$ if $u \geq \sqrt{\ln(n)}$ $O\left(u^{-\beta}n\log\frac{n}{u^2}\right)$ if $u \leq \sqrt{\ln(n)}$
$= 1$	$O(n\log(u))$ if $u \geq \sqrt{\ln(n)}$ $O\left(\frac{n}{u^2}\log\left(\frac{n}{u^2}\right)\log(u)\right)$ if $u \leq \sqrt{\ln(n)}$	$O(nu\log(u))$ if $u \geq \sqrt{\ln(n)}$ $O\left(\frac{n}{u}\log\left(\frac{n}{u^2}\right)\log(u)\right)$ if $u \leq \sqrt{\ln(n)}$
$(1,2)$	$O(n)$ if $u \geq \ln^{\frac{1}{3-\beta}}(n)$ $O\left(\frac{n}{u^{3-\beta}}\log\left(\frac{n}{u^2}\right)\right)$ if $u < \ln^{\frac{1}{3-\beta}}(n)$	$O(nu^{2-\beta})$ if $u \geq \ln^{\frac{1}{3-\beta}}(n)$ $O\left(\frac{n}{u}\log\left(\frac{n}{u^2}\right)\right)$ if $u < \ln^{\frac{1}{3-\beta}}(n)$
$= 2$		$O(n\log(u))$ if $u \geq \ln^{\frac{1}{3-\beta}}(n)$ $O\left(\frac{n\log(u)}{u^{3-\beta}}\log\left(\frac{n}{u^2}\right)\right)$ if $u < \ln^{\frac{1}{3-\beta}}(n)$
$(2,3)$		$O(n)$ if $u \geq \ln^{\frac{1}{3-\beta}}(n)$ $O\left(\frac{n}{u^{3-\beta}}\log\left(\frac{n}{u^2}\right)\right)$ if $u < \ln^{\frac{1}{3-\beta}}(n)$
$= 3$	$O(n\log\log(u))$ if $u \geq n^{\frac{1}{\ln\ln(n)}}$ $O\left(\frac{n}{\log(u)}\log\left(\frac{n}{u^2}\right)\right)$ if $u < n^{\frac{1}{\ln\ln(n)}}$	$O(n\log\log(u))$ if $u \geq n^{\frac{1}{\ln\ln(n)}}$ $O\left(\frac{n}{\log(u)}\log\left(\frac{n}{u^2}\right)\right)$ if $u < n^{\frac{1}{\ln\ln(n)}}$
> 3	$O(n\log(n))$	$O(n\log(n))$

Table 3 – *Upper bounds on the expected number of iterations and expected number of fitness evaluations for different values of β and u. The last column is calculated by Wald's equation in the same manner as in Theorem 29*

3) By the previous condition we have

$$\sum_{t=1}^{+\infty} E[|\lambda_t| \cdot \mathbb{1}_{\{T_I \geq t\}}] = \sum_{t=1}^{+\infty} \Pr[T_I \geq t]E[\lambda_t] = E[\lambda]E[T_I],$$

since for all $t \in \mathbb{N}$ we have $E[\lambda_t] = E[\lambda]$. By Theorem 28 and Lemma 6, both $E[\lambda]$ and $E[T_I]$ are finite, hence their product is finite as well.

4) By Theorem 28 $E[T_I]$ is finite.

Thus, by Wald's inequality we have

$$E[T_F] = E[T_I]E[2\lambda_t].$$

By Theorem 28 we have $E[T_I] = O(n)$ and by Lemma 6 we have $E[\lambda] = \Theta(1)$. Hence, we conclude

$$E[T_F] = O(n) \cdot \Theta(1) = O(n). \qquad \square$$

Although we are mostly interested in $\beta \in (2,3)$ and reasonably high upper limit u, a reader might find it interesting to see the upper bounds for the runtimes yielded by different parameters values.

For this reason we show the estimates for $E[T_I]$ and $E[T_F]$ for a wider range of parameters values in Table 3. We omit the proofs, since they generally imitate the proofs of Theorems 28 and 29.

In the proofs of Theorems 28 and 29 we aimed at delivering only asymptotical upper bounds disregarding the leading constant in order not to reduce the readability of the paper. However, for the complete picture, without proof we estimate the leading constant derived from our arguments.

Recall that $C(\beta) = \frac{1}{12}C_{\beta,u}C'$. From the proof of Lemma 7 in [33] we can show that C' which is used in Lemma 70 is at least $\frac{1}{e}(1 - \exp(-\exp(-\frac{3}{2}))) \approx 0.0735$. For any upper bound $u = \omega(1)$ we also have $C_{\beta,u} \approx \beta - 1$. Hence, we estimate the upper bound on the leading constant.

$$\frac{1}{C(\beta)}\left(1 + \frac{2}{3-\beta}\right) \approx \frac{12(5-\beta)}{(3-\beta)(\beta-1)C'} \approx 164\frac{(5-\beta)}{(3-\beta)(\beta-1)}.$$

Taking into account the leading constant hidden in Lemma 6, which is $\frac{\beta-1}{\beta-2}$ if $\beta > 2$, we estimate the upper bound on the leading constant for $E[T_F]$ delivered by Theorem 29 as

$$328\frac{(5-\beta)}{(3-\beta)(\beta-2)}.$$

Lower Bound

In this section we prove the tightness of our upper bounds by showing a lower bound of $\Omega(n)$ fitness evaluations for the runtime of the heavy-tailed $(1 + (\lambda, \lambda))$ GA on ONEMAX. This is a special case of a deeper result [131], which showed the same lower bound for all comparison-based algorithms (which the $(1 + (\lambda, \lambda))$ GA is). For the readers' convenience, we give an elementary proof as well.

Theorem 30. *The expected runtime of the heavy-tailed $(1 + (\lambda, \lambda))$ GA with parameter $\beta \in \mathbb{R}$ and any upper limit $u \in \mathbb{N}$ on the ONEMAX function is at least $\Omega(\frac{n}{E[\lambda]})$ iterations, where $E[\lambda]$ is estimated as in Lemma 6, and $\Omega(n)$ fitness evaluations.*

Proof. The progress in one iteration cannot be greater than the number ℓ of bits which we flip in each mutant, since we cannot obtain more than ℓ new one-bits in the winner x' of the mutation phase. Therefore, after we have sampled λ, the expected progress is

$$E[f(y) - f(x) \mid \lambda] \leq E[\ell \mid \lambda] = \lambda.$$

The expected progress in one iteration thus is

$$E[f(y) - f(x)] = \sum_{i=1}^{u} \Pr[\lambda = i] E[f(y) - f(x) \mid \lambda = i] \leq E[\lambda].$$

Let x_0 be the initial individual. Since it is chosen uniformly at random, its expected fitness is $E[f(x_0)] = \frac{n}{2}$. Hence, by the additive drift theorem [81] the expectation of the number of iterations T_I before the algorithm finds the optimum is at least

$$E[T_I] \geq \frac{n - E[f(x_0)]}{E[\lambda]} = \frac{n}{2E[\lambda]}.$$

Now we can use Wald's equation as we did in the proof of Theorem 29. We obtain

$$E[T_F] = E[T_I]E[2\lambda] \geq \frac{n}{2E[\lambda]} \cdot 2E[\lambda] = n. \qquad \square$$

4.2.2 Experiments

In order to estimate the leading constant in the runtime of the heavy-tailed $(1 + (\lambda, \lambda))$ GA with a heavy-tailed choice of λ and to compare it with its natural competitors on more practical

problems we performed a series of experiments. We compared our algorithm with standard mutation-based algorithms, namely randomized local search (RLS) and the $(1+1)$ EA with a standard bit mutation, as well as with the $(1+(\lambda,\lambda))$ GA with controlled λ according to the one-fifth rule [31]. We have also considered the version of the $(1+(\lambda,\lambda))$ GA with the one-fifth rule with an upper limit of $2\ln(n+1)$ on the value of λ, introduced in [11], since it showed a much better performance on the MAX-3SAT problem than without this upper limit. In all the adaptive versions of the $(1+(\lambda,\lambda))$ GA, the initial value of λ is set to 1.

Since according to our theoretical results the runtime is linear for all $\beta \in (2,3)$, we were also interested in finding the most appropriate value from this interval. We tried the values of $\beta = 2.1, 2.3, 2.5, 2.7$ and 2.9.

For all runs we slightly modified the algorithms to avoid counting absolutely useless fitness evaluations. The particular changes are as follows.

- In the $(1+1)$ EA, if the standard bit mutation flips zero bits, then we continue resample offspring until one or more bits are flipped.
- In all versions of the $(1+(\lambda,\lambda))$ GA, we (re-)sample ℓ until $\ell \neq 0$. In the crossover phase, the attempts to sample an individual identical to the parent x are repeated (without evaluating the fitness of a copy), and the attempts to sample the individual identical to the mutation-best offspring x' do not count towards the number of fitness evaluations. Additionally, x' also participates in the selection of the best among x and the crossover results $y^{(i)}$. When there is a tie, then the crossover winner has a higher priority than x'.

We consider these natural modifications instead of the original algorithms in this section, since we are sure that anyone implementing these algorithms for solving practical problems would do the same. For a practitioner it does not make sense to waste fitness evaluations on individuals which are identical to their parents, while in theoretical works these are often counted since constant factors are often ignored. We note that similar modifications of algorithms were called *practice-aware* in [115]. We note that there are much more ways to tune the runtime of the $(1+(\lambda,\lambda))$ GA in a practical application, see, e.g., [76]. In contrast to the modifications described above, for these it is not clear to what extent they are useful in general or only for particular problems. For this reason, we did not consider them in our experimental study.

Clearly our theoretical results from Section 4.2.1 apply to these mildly modified algorithms. For the upper bounds it is enough to note that by resampling identical individuals and by having x' participate in the selection, the probability to have a progress in one iteration only increases. Thus, repeating the arguments from Theorem 28 we obtain the same upper bound on the expected number of iterations. Since our implementation does not affect the choice of λ, its expected value $E[\lambda]$ stays the same. The cost of one iteration is at most 2λ (but can be smaller). Thus, by Wald's equation we obtain the same upper bound on the expected number of fitness evaluations as in Theorem 29. For the lower bound we use the same arguments as in Theorem 30, with the only change that since we cannot choose $\ell = 0$, we have

$$E[\ell \mid \lambda] = \frac{\lambda}{1-\left(1-\frac{1}{\lambda}\right)^{\lambda}} \leq \frac{\lambda}{1-\frac{1}{e}},$$

which still gives us a lower bound of $\Omega(n)$ fitness evaluations.

The experiments were performed on the OneMax function and on random satisfiable instances of the MAX-3SAT problem, that is, the problem of maximizing the number of satisfied clauses in a Boolean formula represented in the conjunctive normal form. The second problem was chosen for two reasons. First, it is a more practical problem than OneMax, second, there are

already theoretical and empirical results for the $(1 + (\lambda, \lambda))$ GA on this function (see [11]). For this problem, the number of clauses was chosen to be $4n \ln n$ for the number of variables n. An all-ones bit string is assumed to be a planted optimal solution; this is without loss of generality, as all considered algorithms are unbiased. For each clause, three participating variables and their signs (i.e. whether it is inverted or not) are sampled uniformly and independently until this clause is satisfied by the planted solution. Note that these are easy instances of the MAX-3SAT problem, so the presented results on this problem should not be considered as if the proposed algorithms are competitive in solving this problem in general. However, these instances have a lower fitness-distance correlation, which makes them harder in particular for the $(1 + (\lambda, \lambda))$ GA.

In our experiments we chose the problem sizes n to be powers of two, so that the asymptotic behavior of the algorithms is easier to investigate visually. For OneMax, we limit the problem size to 2^{22}, and for MAX-3SAT, the upper limit is 2^{15}. These sizes were derived from the affordable computational times. To allow also these high problem sizes, we used incremental fitness evaluations to save computational time without affecting any other aspect of the experiment. We did not reach the size of 2^{20} on MAX-3SAT as in [11], because the incremental fitness evaluations have a weaker impact on algorithms with fast mutation. For each algorithm, each problem setting, and each problem size, 100 independent runs were performed. For the MAX-3SAT problem, a new random instance was created for each run.

The results are shown in Figures 10 and 11. In both figures the x-axis indicates the problem size in a logarithmic scale, and the y-axis indicates the ratio of the runtime to the problem size. In this visualization a linear runtime results in a horizontal plot and any runtime in $\Theta(n \log n)$ gives a linearly increasing plot.

In Figure 10 we show the results of the runs on the OneMax function. If we do not consider $\beta = 2.1$, which turns out to be too small (and therefore gives a too large expected value of λ), then all versions of the heavy-tailed $(1 + (\lambda, \lambda))$ GA start outperforming the $(1 + 1)$ EA already at population size $n = 2^{10}$ and then outperform RLS at $n = 2^{20}$ or earlier. Recalling the discussion after the proof of Theorem 28 we note that our estimate of the leading constant in the runtime is way too pessimistic, otherwise we would have no chance to outperform RLS on these problem sizes.

The one-fifth rule shows a much better performance and yields a runtime of the $(1 + (\lambda, \lambda))$ GA which is close to linear already at $n = 2^{10}$, independently of the upper bounds on the choice of λ. The plots for the heavy-tailed choice of λ do not look horizontal, but they show a strongly marked tendency that they will do it at larger population sizes. The runtimes for all β except $\beta = 2.1$ are quite well concentrated, as well as the runtimes of the $(1 + (\lambda, \lambda))$ GA with the one-fifth rule, in contrast to the runtimes of the $(1 + 1)$ EA and RLS. We have no results for $\beta = 2.1$ for population sizes $n \geq 2^{21}$ and for $\beta = 2.3$ for $n \geq 2^{22}$, since they were too expensive (in terms of computational resources) and most likely not too insightful.

Regarding the runtimes of the $(1 + (\lambda, \lambda))$ GA with fixed values of λ one can see that the heavy-tailed $(1 + (\lambda, \lambda))$ GA performs roughly similarly. With $\beta = 2.5$ it manages to outperform the optimal and close-to-optimal fixed λ (which were computed in [11]) when the problem size $n \geq 2^{20}$. Note that the theoretically asymptotically optimal $\lambda = \sqrt{\frac{\ln(n) \ln\ln(n)}{\ln\ln\ln(n)}}$ (delivered in [31]) is outperformed by different static values of λ at these population sizes.

Figure 11 shows the results of the experiments on the MAX-3SAT problem. As previously shown in [11], large values of λ can be harmful. For this reason, the $(1 + (\lambda, \lambda))$ GA with the unbounded one-fifth rule is outperformed already by the simple $(1 + 1)$ EA. The authors of [11] proposed to limit the value which λ can take by $2 \ln(n + 1)$, which helped to outperform RLS

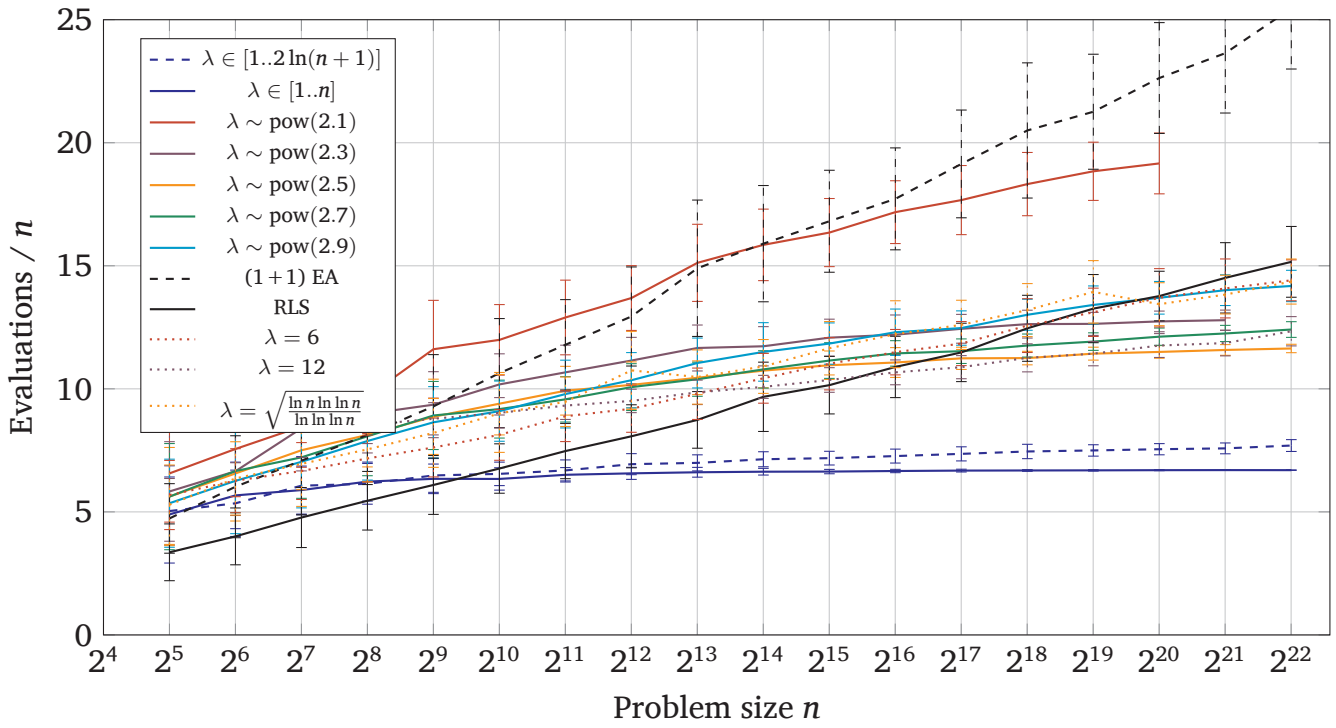


Figure 10 – *Mean runtimes and their standard deviation of different algorithms on* OneMax *benchmark problem. By* $\lambda \in [1..u]$ *we denote the self-adjusting parameter choice via the one-fifth rule in the interval* $[1..u]$*. The indicated confidence interval for each value* X *is* $[E[X] - \sigma(X), E[x] + \sigma(X)]$*, where* $\sigma(X)$ *is the standard deviation of* X

on this problem. As we see in Figure 11, the heavy-tailed $(1 + (\lambda, \lambda))$ GA is quite efficient even without an upper limit on λ, except when $\beta = 2.1$. All other values of β managed to outperform the $(1 + 1)$ EA and the $(1 + (\lambda, \lambda))$ GA with the one-fifth rule without the upper limit on λ (however, for $\beta = 2.3$ the advantage is not very clear), but none of them outperformed RLS or the $(1 + (\lambda, \lambda))$ GA with the one-fifth rule with a logarithmic cap on λ. The runtimes of all algorithms are super-linear according to the plots.

Summing up, from the results of the experiments we conclude the following three points.

- The heavy-tailed $(1 + (\lambda, \lambda))$ GA performs better than the mutation-based algorithms on OneMax and it is not significantly outperformed on MAX-3SAT.
- It is more universal than the $(1 + (\lambda, \lambda))$ GA with the one-fifth rule, since it works well even without limiting λ. Such limits might be problem-specific, e.g. with a logarithmic limit which is beneficial when solving MAX-3SAT, it may be hard to leave a local optimum with a large basin of attraction.
- The choice of β does not play a big role as long as it is not too close to the borders of the interval $(2, 3)$. Taking β between 2.5 and 2.7 might be a good general recommendation.

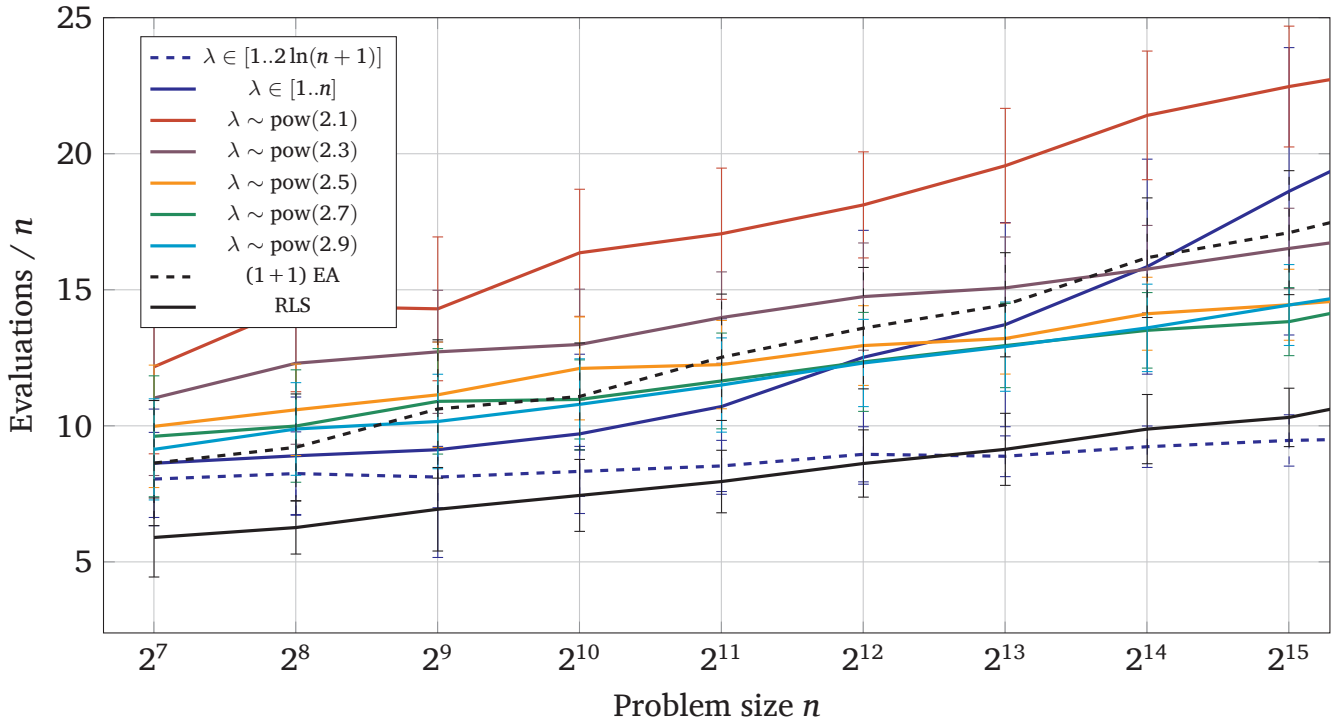


Figure 11 – *Mean runtimes and their standard deviation of different algorithms on MAX-3SAT instances with* $4n\ln(n)$ *clauses. By* $\lambda \in [1..u]$ *we denote the self-adjusting parameter choice via the one-fifth rule in the interval* $[1..u]$*. The indicated confidence interval for each value* X *is* $[E[X] - \sigma(X), E[x] + \sigma(X)]$*, where* $\sigma(X)$ *is the standard deviation of* X

4.3 The Runtime Analysis of the $(1 + (\lambda, \lambda))$ GA on LEADINGONES

What has not been regarded so far, and what is the topic of this section, is how the $(1 + (\lambda, \lambda))$ GA performs on functions which have a weak fitness-distance correlation. The natural first example to regard for such an investigation is the classical LEADINGONES benchmark function. This function is still easy in the sense that it is unimodal, that is, from every search point there is a path to the optimum such that each edge on this path refers to a one-bit flip increasing the fitness. However, the fitness-distance correlation is low, since all bits to the right of the left-most zero have no influence on the fitness.

This immediately suggests that the $(1 + (\lambda, \lambda))$ GA could have some difficulties to optimize LEADINGONES. To make progress when optimizing LEADINGONES, it is necessary to flip the left-most zero-bit ("critical bit"). However, as soon as a one-bit to the left of it is flipped, which is likely when using a large mutation rate, the critical bit has no influence anymore on the fitness. Hence there is no reason why the mutation winner should have a good chance to have the critical bit set to one.

Surprisingly, transforming this intuitive consideration into a rigorous proof turns out to be quite non-trivial (we sketch the difficulties and our solutions before the proofs), but we manage to show that the $(1 + (\lambda, \lambda))$ GA on the LEADINGONES function indeed does not profit from the faster exploration that was aimed at with this algorithm. On the positive side, we can also

show that the $(1+(\lambda,\lambda))$ GA is still asymptotically as efficient as many classical evolutionary algorithms.

The main result of this section is that the runtime of the $(1+(\lambda,\lambda))$ GA with standard parameter setting $p=\lambda/n$ and $c=1/\lambda$ for any population size $\lambda \leq n/2$ finds the optimum of the n-dimensional LEADINGONES function in an expected number of $\Theta(n^2/\lambda)$ iterations, which is equivalent to an expected number of $\Theta(n^2)$ fitness evaluations. The latter bound on the number of fitness evaluations is also valid when the value of λ is chosen in an arbitrary dynamic manner, including the choice from a power-law distribution with $u \leq \frac{n}{2}$. We note that values for λ that are greater than $n/2$ in the standard parameterization do not make much sense for the $(1+(\lambda,\lambda))$ GA, since the mutation rate p is set to $p=\lambda/n$. Hence a λ-value greater than $n/2$ gives a mutation rate larger than $1/2$, which is generally considered to be ill-natured.

In the light of the runtimes of the natural competitors described in Subsection 1.4.3 we see that the $(1+(\lambda,\lambda))$ GA also in the absence of a strong fitness-distance correlation can be a competitive optimization algorithm. It may lose its particularly efficient faster exploration, but even then it competes well with the classical evolutionary algorithms.

As a side result we prove a variant of the additive drift theorem for lower bounds that comes with a tail bound (Theorem 31 with a more general Theorem 32). Such a result was given previously in [92], however, it used a notion of sub-Gaussian processes and was not easy to apply in this work. The differences with those previous results are discussed in details in Subection 4.3.1.

4.3.1 Additive Drift with Tail Bounds

In our proofs we face the need to give an upper bound on the runtime of some process that has a drift of at most some ε towards its goal. Namely, we want to argue that the difference between best-so-far fitness (which can be greater than the current fitness $f(x)$ if it was obtained in the mutation phase) and $f(x)$ does not exceed $\frac{n}{4}$ for long enough with high probability. For these reason, we prove the following theorem.

Theorem 31 (Additive drift with tail bounds)**.** *Let X_t, $t=0,1,2,\dots$ be integer-valued random variables that describe some stochastic process over some state space. Let $X_0=a$ and let T be the first time when X_t is at least some $b>a$. If there exist $\delta>0$ and $r>0$ such that for all $j\in\mathbb{N}$[10] we have*

$$\Pr[X_{t+1}-X_t=j \mid X_0,\dots,X_t] \leq \frac{r}{(1+\delta)^j},$$

then with

$$\varepsilon := 2r\frac{2+\delta}{\delta\ln(1+\frac{\delta}{2})}$$

we have

$$\Pr\left[T \leq \frac{(b-a)}{2\varepsilon}\right] \leq \max\left\{4, \frac{2\delta}{r(2+\delta)}\right\}\cdot\frac{(b-a)}{2\varepsilon}\left(1+\frac{\delta}{2}\right)^{-\frac{b-a}{2}}.$$

[10]Here and further in the paper by the set of natural numbers $\mathbb{N}$ we mean the set of all positive integers $\{1,2,\dots\}$.

Note that the random variable X_t considered in this theorem can have a positive drift of at most

$$\begin{aligned}\Delta_X &\le \sum_{j=1}^{+\infty} j \Pr[X_{t+1} - X_t = j \mid X_0, \dots, X_t] \\ &\le \sum_{j=1}^{+\infty} j \frac{r}{(1+\delta)^j} = \frac{r(1+\delta)}{\delta^2}.\end{aligned}$$

Hence, ε, as defined in the theorem, is at most by factor $\frac{2\delta(2+\delta)}{(1+\delta)\ln(1+\frac{\delta}{2})}$ greater than the drift of X_t.

To prove Theorem 31 we first formulate and prove the following more general theorem.

Theorem 32. *Let X_t, $t = 0, 1, 2, \dots$ be a real-valued random process. Let $X_0 = a$ and let T be the minimal t such that $X_t \ge b$ for some $b > a$. Let $Y_t = X_t - \varepsilon t$ with some $\varepsilon > 0$. If there exist $\gamma > 0$ and $p > 1$ such that*

$$E[e^{\gamma(Y_{t+1}-Y_t)} \mid X_0 \dots X_t] \le 1 - \frac{1}{p},$$

then we have

$$\Pr\left[T \le \frac{(b-a)}{2\varepsilon}\right] \le \frac{(b-a)p}{2\varepsilon} \exp\left(-\frac{\gamma(b-a)}{2}\right).$$

Note that in Theorem 32 one should choose ε which is greater than the drift of X_t. Otherwise, we have

$$E[e^{\gamma(Y_{t+1}-Y_t)} \mid X_0 \dots X_t] \ge E[\gamma(Y_{t+1} - Y_t) \mid X_0 \dots X_t] + 1 \ge 1.$$

The main argument in the proof of Theorem 32 is the application of the negative drift theorem (Theorem 1).

Proof of Theorem 32. Let T_Y be the first time when Y_t becomes at least $\frac{b+a}{2}$. If $t \le \frac{b-a}{2\varepsilon}$ and $Y_t < \frac{b+a}{2}$ then $X_t = Y_t + \varepsilon t < \frac{b+a}{2} + \frac{b-a}{2} = b$. Therefore,

$$\Pr\left[T_X \le \frac{(b-a)}{2\varepsilon}\right] \le \Pr\left[T_Y \le \frac{(b-a)}{2\varepsilon}\right].$$

To apply Theorem 1 to Y_t we compute

$$\begin{aligned}D &= \max_{t\in\mathbb{N}} \left\{1, E[e^{\gamma(Y_{t+1}-a)} \mid Y_t \le a]\right\} \\ &\le \max_{t\in\mathbb{N}} \left\{1, E[e^{\gamma(Y_{t+1}-Y_t)} \mid Y_t \le a]\right\} \le \max\left\{1, 1 - \frac{1}{p}\right\} = 1.\end{aligned}$$

Finally, by Theorem 1 with $L := \frac{n}{2\varepsilon}$ and $d := \frac{b+a}{2} - a = \frac{b-a}{2}$ we conclude

$$\begin{aligned}\Pr\left[T_Y \le \frac{(b-a)}{2\varepsilon}\right] &\le \frac{(b-a)}{2\varepsilon} \cdot 1 \cdot p \cdot \exp\left(-\frac{\gamma(b-a)}{2}\right) \\ &= \frac{(b-a)p}{2\varepsilon} \exp\left(-\frac{\gamma(b-a)}{2}\right).\end{aligned}$$

□

Finally we are ready to prove the main result of this section.

Proof of Theorem 31. In order to apply Theorem 32 set $\gamma := \ln(1+\frac{\delta}{2})$ and compute

$$\begin{aligned} E[e^{\gamma(Y_{t+1}-Y_t)}] &= E[e^{\gamma(X_{t+1}-X_t-\varepsilon)}] = e^{-\gamma\varepsilon}E[e^{\gamma(X_{t+1}-X_t)}] \\ &\leq e^{-\gamma\varepsilon}\left(\Pr[X_{t+1}-X_t\leq 0] + \sum_{j=1}^{+\infty}\frac{e^{\gamma j}r}{(1+\delta)^j}\right) \\ &\leq e^{-\gamma\varepsilon}\left(1+r\frac{2+\delta}{\delta}\right). \end{aligned} \tag{47}$$

From the definitions of ε and γ we have $r\frac{2+\delta}{\delta} = \frac{\gamma\varepsilon}{2}$. Therefore, by Lemma 16 and by Eq. (47) we have

$$\begin{aligned} E[e^{\gamma(Y_{t+1}-Y_t)}] &\leq \exp\left(-2r\frac{2+\delta}{\delta}\right)\left(1+r\frac{2+\delta}{\delta}\right) \\ &\leq \max\left\{1-\frac{1}{4}, 1-r\frac{2+\delta}{2\delta}\right\}. \end{aligned} \tag{48}$$

We define $p := \max\{4, \frac{2\delta}{r(2+\delta)}\}$. By Eq. (48) we have

$$E[e^{\gamma(X_{t+1}-X_t)}] \leq 1-\frac{1}{p}. \tag{49}$$

Therefore, by Theorem 32 we obtain

$$\Pr\left[T_Y \leq \frac{(b-a)}{2\varepsilon}\right] \leq \max\left\{4, \frac{2\delta}{r(2+\delta)}\right\}\cdot\frac{(b-a)}{2\varepsilon}\left(1+\frac{\delta}{2}\right)^{-\frac{b-a}{2}}.$$

□

Comparison with Existing Tools

We now compare our results from Subsection 4.3.1 with two similar results presented in [92]. The first of them is the following theorem.

Theorem 33 (Theorem 1 in [92]). *Let X_t, $t = 0,1,2,\ldots$ be real-valued random variables with finite expectation and let $X_0 = a$. Let T be the first moment in time when X_t exceeds some $b > a$. Suppose there exist $\varepsilon, c > 0$ such that for all t we have*

1) $E[X_{t+1} - X_t \mid X_0,\ldots,X_t, T > t] \leq \varepsilon$ and

2) $|X_t - X_{t+1}| \leq c$.

Then for all $s \leq \frac{b-a}{2\varepsilon}$ we have

$$\Pr[T < s] \leq \exp\left(-\frac{(b-a)^2}{8c^2 s}\right).$$

Unfortunately, this result is not strong enough for our purposes, since we consider a process that can perform larger jumps. There is a decent chance that a jump of order $\Omega(\log(b-a))$ happens in a relatively small number of iterations. Thus, we could only use Theorem 33 with a super-constant c, which would not yield the desired bound on the failure probability.

The second related result from [92] uses the notion of *sub-Gaussian processes*. A random process X_t is called (c, δ)-*sub-Gaussian* if for all $t \in \mathbb{N}$ and for all $\gamma \in [0, \delta]$ we have

$$E[e^{\gamma(X_{t+1}-X_t)} \mid X_0, \ldots, X_t] \leq \exp\left(\frac{\gamma^2 c}{2}\right).$$

For such processes the following result was shown in [92].

Theorem 34 (Theorem 14 in [92])**.** *Suppose X_t has a drift of at most $\varepsilon > 0$ and $(X_t - \varepsilon t)$ is (c, δ)-sub-Gaussian. Let X_0 be $a \in R$ and let T be the first moment in time when X_t exceeds some $b > a$. Then for all $s \leq \frac{b-a}{2\varepsilon}$ we have*

$$\Pr[T \leq s] \leq \exp\left(-\frac{b-a}{4} \min\left(\delta, \frac{b-a}{2cs}\right)\right).$$

Generally this theorem satisfies our needs, and could have been used instead of Theorem 32 in the proof of Theorem 31. However, in order to do so we would have to prove that the process $X_t - \varepsilon t$ under conditions of Theorem 31 is (c, δ)-sub-Gaussian. To do so we would have to find some $c \in \mathbb{R}$ and $\delta > 0$ such that for all $\gamma \in [0, \delta]$ we had

$$E[e^{\gamma(X_{t+1}-X_t-\varepsilon)} \mid X_0, \ldots, X_t] \leq \exp\left(\frac{\gamma^2 c}{2}\right). \tag{50}$$

Compare this to the statement which should be proved in order to use Theorem 32.

$$E[e^{\gamma(X_{t+1}-X_t-\varepsilon)} \mid X_0, \ldots, X_t] \leq 1 - \frac{1}{p}. \tag{51}$$

The main difficulty that one can face when applying Theorem 34 is proving that Eq. (50) holds for all values of $\gamma \in [0, \delta]$. At the same time, to use Theorem 32 it is enough to prove that Eq. (51) holds for only one particular value of γ. This might be crucial, when the left part of Eq. (50) is not monotonic in γ.

Theorem 10 in [92] proves that all processes with step size which is dominated by some random variable with geometric distribution are (c, δ)-sub-Gaussian and gives exact values for c and δ which depend on the parameters of that geometric distribution. However, this theorem does not satisfy our needs, since we consider processes which can make large steps, but only in direction which is opposite to their goal.

Another reason why we formulated and proved Theorem 32 is that for any (c, δ)-sub-Gaussian process with $c < 0$ Theorem 34 bounds the probability $\Pr[T < s]$ with a value which is greater than one. This can be overcome by the observation that any (c, δ)-sub-Gaussian process is (c', δ)-sub-Gaussian for all $c' > c$. For this reason we can consider any (c, δ)-sub-Gaussian process with negative c as a $(\frac{b-a}{2\delta s}, \delta)$-sub Gaussian, so that Theorem 34 yields

$$\Pr[T \leq s] \leq \exp\left(-\frac{(b-a)\delta}{4}\right).$$

This bound can be made more precise for the processes X_t, $t = 0, 1, 2, \ldots$ such that $X_t - \varepsilon t$ has a truly negative drift, that is, when the process satisfies conditions of Theorem 32. Notice that in this setting, if it is possible also to show that Eq. (51) holds for all smaller positive γ (which means that the process $X_t - \varepsilon t$ is $(0, \gamma)$-sub-Gaussian), then the bound yielded by Theorem 34 is

$$\Pr[T \leq s] \leq \exp\left(-\frac{n\gamma}{4}\right),$$

which is asymptotically greater than the bound delivered by Theorem 32 with any p which is polynomial in n.

4.3.2 Lower Bound

The main result of this section is the following theorem.

Theorem 35. *If $\lambda \leq \frac{n}{2}$, the expected runtime of the $(1 + (\lambda, \lambda))$ GA on LeadingOnes is $\Omega(\frac{n^2}{\lambda})$ iterations or $\Omega(n^2)$ fitness evaluations.*

When proving this result, we have to overcome a couple of technical challenges. One of these is that it is hard to argue with the fitness of the current solution x. A common argument in the runtime analysis for the LeadingOnes function is that when the current-best search point x has a fitness of i, then the first i bits of x are one, the next bit is zero, and all other bits are independently and uniformly distributed on $\{0, 1\}$. This argument already appeared in the seminal runtime analysis paper [67]. It gives a very good understanding of the search process, which allows very precise analyses of the expected runtime [7, 125] or even the distribution of the runtime [54, 26].

The problem in the analysis of the $(1 + (\lambda, \lambda))$ GA is that not always the best search point ever generated survives as parent individual. It may well happen that the mutation phase creates a search point with high fitness i^+ (which then becomes the mutation winner), but that then the crossover phase only creates individuals with lower fitness. If one of these becomes the next parent, say with fitness i, then bit $i+1$ surely is zero, but the bits $i+2, \ldots, i^+ + 1$ are not uniformly distributed in $\{0, 1\}$, since they were influenced by the selection in the mutation phase.

The usual way to overcome this problem, done first in [55, proof of Theorem 10] and subsequently in [94, 52, 121, 128, 38], is to argue that such bits relatively quickly approach the uniform distribution. More precisely, the distance to the uniform distribution reduces by a factor of $(1 - 2/n)$ each round. Hence $\Theta(n)$ iterations, with implicit constant large enough, suffice to reduce the distance to any small constant. This convergence speed is usually enough when working with standard bit mutation with the classical mutation rate of $1/n$, since it takes $\Theta(n)$ iterations to find the next fitness improvement.

In our situation, unfortunately, the more aggressive way of mutation together with the fact that offspring are generated in parallel can imply that a fitness improvement is found much faster than the time needed to let these bits converge to a near-uniform distribution. For this reason, we do not see how to use the argument just sketched.

We overcome this problem by not regarding the progress with respect to the fitness of the current search point x, but with regard to the best fitness ever seen in the run so far. Hence we denote by $f_{\max}(t)$ the maximum fitness among all search points sampled in the first $t-1$ iterations

(so that $f_{\max}(t)$ and $f(x)$ are the fitnesses of the best-so-far and the current search point at the start of iteration t). As discussed, $f_{\max}(t) = n$ does not mean that the algorithm has reached the optimum. However, the converse is true, and this is what we need. As soon as the algorithm reaches the optimum, $f_{\max}(t)$ must be n. For this reason, a lower bound on the time when $f_{\max}(t)$ hits n is a lower bound for the runtime as well.

In simple words, our main proof argument will be that when $f_{\max}(t)$ is in the interval $[\frac{1}{2}n, \frac{3}{4}n]$, its value increases only by $O(\lambda/n)$ in expectation at each iteration, and that this implies that the expected number of iterations necessary to obtain an $f_{\max}(t)$-value larger than $\frac{3}{4}n$ is $\Omega(n^2/\lambda)$. We encounter some more technical challenges on the way, but to ease reading we shall discuss them before the proof in which they show up.

The Probability of Successful Mutation

If we denote the current fitness $f(x)$ by i, then in order to generate a better individual (either in the mutation phase or in the crossover phase) it is necessary that the $i+1$-st bit of the mutation winner x' is one. We first analyze the probability of this event. We distinguish three possible outcomes of the mutation phase that lead to this.

1) $f(x') < f(x)$ and the $i+1$-st bit is flipped in x'.
2) $f(x') > f(x)$ and $f(x') \leq f_{\max}(t)$.
3) $f(x') > f_{\max}(x)$.

In this subsection we study these three cases separately, in the order listed above. Note that we do not consider the case when $f(x') = f(x)$, since it implies that the $i+1$-st bit is zero in x'.

Lemma 72. *Let i be the current fitness $f(x)$. If $i \leq \frac{3n}{4}$, the probability p_M^- that $f(x') < f(x)$ and the $i+1$-st bit is flipped in x' is at most $\frac{4\lambda}{n}$.*

Proof. If the algorithm has chosen mutation strength ℓ in the mutation phase and $f(x') = j < f(x)$, then the $j+1$-st bit is surely flipped in x' and the other $\ell-1$ flipped bits are somewhere in positions $[j+2..n]$, distributed there uniformly at random. Hence, the probability that the i-th bit is flipped in x' is $\frac{\ell-1}{n-j-1}$. Therefore, the probability that x' is worse than x, but the $i+1$-st bit is flipped, is

$$p_M^- = \sum_{k=2}^{n} \Pr[\ell = k] \sum_{j=0}^{\min\{f(x)-1, n-k\}} \Pr[f(x') = j \mid \ell = k] \frac{k-1}{n-j-1}.$$

Since we assume that $i \leq \frac{3n}{4}$, we have $j \leq i-1 < \frac{3n}{4}$ and thus $n-j-1 \geq \frac{n}{4}$. Hence, the inner sum can be estimated by

$$\begin{aligned} &\sum_{j=0}^{\min\{i-1, n-k\}} \Pr[f(x') = j \mid \ell = k] \frac{k-1}{n-j-1} \\ &\qquad \leq (k-1) \sum_{j=0}^{\min(i-1, n-k)} \frac{\Pr[f(x') = j \mid \ell = k]}{n/4} \end{aligned}$$

$$< \frac{4(k-1)}{n}.$$

Therefore, we have

$$\begin{aligned} p_M^- &\leq \frac{4}{n}\sum_{k=2}^{n}(k-1)\Pr[\ell = k] \\ &\leq \frac{4}{n}E[\ell - 1] = \frac{4(\lambda-1)}{n} \leq \frac{4\lambda}{n}. \end{aligned}$$

□

Lemma 73. *If $\lambda \leq \frac{n}{2}$, the probability that $f(x') > f(x)$ is at most*

$$\frac{\lambda^2}{n}\exp\left(-\frac{f(x)\lambda}{n}\right).$$

Lemma 74. *If $\lambda \leq \frac{n}{2}$, for all $j \in \mathbb{N}$ we have the probability $p_M(j)$ that $f(x') \geq f_{\max}(t) + j$ is at most*

$$\frac{\lambda^2}{2^{j-1}n}\exp\left(-\frac{f_{\max}(t)\lambda}{n}\right).$$

To prove Lemma 73 and Lemma 74, we use the following auxiliary lemma.

Lemma 75. *If $\lambda \leq \frac{n}{2}$, then for all $m \in [1..\lambda]$ and all $j \in \mathbb{N}$ the probability that $f(x^{(m)}) \geq f_{\max}(t) + j$ is at most*

$$\frac{\lambda}{2^{j-1}n}\exp\left(-\frac{f_{\max}(t)\lambda}{n}\right)$$

and the probability that $f(x^{(m)}) > f(x)$ is at most

$$\frac{\lambda}{n}\exp\left(-\frac{f(x)\lambda}{n}\right).$$

Proof. To obtain a mutant $x^{(m)}$ that is better than x, one needs to flip the bit in position $f(x)+1$, while not flipping a single bit in positions $[1..f(x)]$. To obtain a mutant $x^{(m)}$ of fitness at least $f_{\max}(t)+j$ one also needs to flip those bits in positions $[f(x)+2..f_{\max}(t)+j]$ which are zero. To estimate these probabilities, we recall that by Lemma 1 we can assume that $x^{(m)}$ is obtained from x by flipping all bits independently with probability $\frac{\lambda}{n}$.

For the bits in positions $[1..f(x)]$, the probability that none of them is flipped is $(1-\frac{\lambda}{n})^{f(x)}$. The probability to flip the bit in position $f(x)+1$ is $\frac{\lambda}{n}$. At this point we already have

$$\Pr[f(x^{(m)}) > f(x)] \leq \frac{\lambda}{n}\left(1-\frac{\lambda}{n}\right)^{f(x)} \leq \frac{\lambda}{n}\exp\left(-\frac{f(x)\lambda}{n}\right).$$

For the bits in positions $[f(x)+2..f_{\max}(t)+1]$ we cannot know neither their value, nor their probability to be one or zero. For each such bit the probability that it is a one-bit in $x^{(m)}$ is $q\frac{\lambda}{n} + (1-q)(1-\frac{\lambda}{n})$, where q is the probability of this bit to be a zero-bit in x. Since we

assume that $\lambda \leq \frac{n}{2}$ then $\max\{\frac{\lambda}{n}, (1-\frac{\lambda}{n})\} = (1-\frac{\lambda}{n})$, hence this probability is at most $(1-\frac{\lambda}{n})$. Consequently, the probability that all bits in positions $[f(x)+2..f_{\max}(t)+1]$ are one-bits in $x^{(m)}$ is at most $(1-\frac{\lambda}{n})^{f_{\max}(t)-f(x)}$.

Finally, if $j > 1$, there are $(j-1)$ bits in positions $[f_{\max}(t)+2..f_{\max}(t)+j]$ and they are independently and uniformly distributed in $\{0,1\}$. Thus the probability for each of them to be a one-bit in $x^{(m)}$ is $\frac{1}{2}\frac{\lambda}{n} + \frac{1}{2}(1-\frac{\lambda}{n}) = \frac{1}{2}$. By independence, the probability that they all are one-bits is $\frac{1}{2^{j-1}}$.

In summary, we have

$$\begin{aligned}\Pr[f(x^{(m)}) \geq f_{\max}(t)+j] &\leq \left(1-\frac{\lambda}{n}\right)^{f(x)} \frac{\lambda}{n} \left(1-\frac{\lambda}{n}\right)^{f_{\max}(t)-f(x)} \frac{1}{2^{j-1}} \\ &\leq \frac{\lambda}{2^{j-1}n} \exp\left(-\frac{f_{\max}(t)\lambda}{n}\right).\end{aligned}$$

□

Proof of Lemma 73 and Lemma 74. By the union bound (see Lemma 9) and by Lemma 75 we estimate the probability that $f(x') > f(x)$ as

$$\Pr[f(x') > f(x)] \leq \sum_{m=1}^{\lambda} \Pr[f(x^{(m)}) > f(x)] \leq \frac{\lambda^2}{n} \exp\left(-\frac{f(x)\lambda}{n}\right).$$

And the same argument shows Lemma 74

□

To simplify the following proofs we also state the following upper bound on the probability of large jumps of $f_{\max}(t)$ that trivially follows from Lemma 74.

Corollary 3. *If $\lambda \leq \frac{n}{2}$, then the probability that $f(x') \geq f_{\max}(t) + \frac{n}{16}$ is $e^{-\Omega(n)}$ and the probability that $f(x') \geq f_{\max}(t) + 2\log_2 n$ is at most $\frac{1}{2n}$.*

Proof. By Lemma 74, with $j := \frac{n}{16}$ we have

$$\begin{aligned}\Pr\left[f(x') \geq f_{\max}(t) + \frac{n}{16}\right] &\leq \frac{2\lambda^2}{2^{n/16}n} \exp\left(-\frac{f_{\max}(t)\lambda}{n}\right) \\ &\leq \frac{2\lambda^2}{n} e^{-\Omega(n)} = e^{-\Omega(n)}.\end{aligned}$$

With $j := 2\log_2 n$ we also have

$$\begin{aligned}\Pr[f(x') \geq f_{\max}(t) + 2\log_2 n] &\leq \frac{2\lambda^2}{2^{2\log_2 n}n} \exp\left(-\frac{f_{\max}(t)\lambda}{n}\right) \\ &\leq \frac{2\lambda^2}{n^3} \leq \frac{1}{2n}.\end{aligned}$$

□

Maximal Fitness and Current Fitness

As said earlier, our general proof idea is to show that $f_{\max}(t)$ increases at most by something like $O(\frac{\lambda}{n})$ per iteration (in expectation) and from this deduce a runtime of $\Omega(\frac{n^2}{\lambda})$ iterations. It is not surprising that in the very early stages of the process, when $f_{\max}(t)$ is very small, a larger progress can be observed. For this reason, we shall only regard the phase in which $f_{\max}(t)$ is in the interval $[\frac{1}{2}n, \frac{3}{4}n]$. Surprisingly, even in this regime a too large (super-constant) progress $f_{\max}(t+1) - f_{\max}(t)$ is possible in certain situations, namely when $f(x)$ is very small (not greater than some constant). We first give an example and then show that such situations arise not too often.

Let $f(x)$ be zero and let $f_{\max}(t)$ be $\frac{n}{2}$. In this case the leading bit of x is zero, but we know nothing about the other bits in the first half of x, including their probabilities to be one or zero. So in the most optimistic case, they all are one. Without giving details we note that the probability that the first bit is flipped in x' is approximately $1 - (1 - \frac{\lambda}{n})^\lambda = \Theta(\min\{1, \frac{\lambda^2}{n}\})$.

In the crossover phase we have a probability of $\frac{1}{\lambda}$ to take the leading bit from x' into a fixed offspring. At the same time, the number of bits that are flipped in x' among the first $\frac{n}{2}$ bits is λ on average, so the probability to repair them all in the crossover phase by taking them from x is around $(1 - \frac{1}{\lambda})^\lambda \approx e^{-1}$. Note that having the first $\frac{n}{2}$ bits equal to one gives us fitness that is at least $\frac{n}{2} + 1$. Therefore, the probability that we create an offspring with fitness strictly greater than $f_{\max}(t)$ in the crossover phase is approximately $1 - (1 - \frac{1}{2e\lambda})^\lambda \approx 1 - e^{-\frac{1}{2e}}$, that is a positive constant.

These observations, although informal, lead to a drift of order $\Theta(\frac{\lambda^2}{n})$. For this reason we now argue that typically $f_{\max}(t)$ is not extremely far from $f(x)$, which excludes the given example from the possible scenarios.

Lemma 76. *If $\lambda < \frac{n}{2}$ then there exists some $\tau = \Theta(\frac{n^2}{\lambda})$ such that the difference $f_{\max}(t) - f(x)$ does not exceed $\frac{n}{4}$ in first τ iterations with probability $1 - e^{-\Omega(n)}$.*

Proof. Let $X_t := f_{\max}(t) - f(x)$, where x is the parent individual from iteration t. X_t can increase only in the mutation phase, since any individual created in the crossover phase that is better than the current x can only decrease X_t. Therefore, the probability that X_t increases by at least j in one iteration is at most the probability that we create x' of fitness $f_{\max}(t) + j$. By Lemma 74, for all $j \in \mathbb{N}$, this probability is at most $\frac{\lambda^2}{2^{j-1}n} \exp(-\frac{f_{\max}(t)\lambda}{n})$.

Let t_0 be the first iteration when $f_{\max}(t) > \frac{n}{16}$. Before this iteration we have $X_t \le \frac{n}{16}$, since $X_t \le f_{\max}(t) \le \frac{n}{16}$. The probability that Y_t increases drastically in iteration $t_0 - 1$ is small, since by Corollary 3 we have

$$\Pr\left[X_{t_0} \ge X_{t_0-1} + \frac{n}{16}\right] \le \Pr\left[f(x') \ge f_{\max}(t_0 - 1) + \frac{n}{16}\right] = e^{-\Omega(n)}.$$

Consequently,

$$\Pr\left[X_{t_0} \ge \frac{n}{8}\right] = e^{-\Omega(n)}.$$

We aim to apply Theorem 31 to X_t, counting t_0 as the initial iteration. For this purpose we define

$$r := \frac{32\lambda}{en} \ge \frac{2\lambda^2}{n} e^{-\frac{\lambda}{16}} \ge \frac{2\lambda^2}{n} \exp\left(-\frac{f_{\max}(t)\lambda}{n}\right),$$

where the first inequality follows from Lemma 14, and $\delta := 1$ so that by Lemma 74 we had

$$\Pr\left[X_{t+1} - X_t \geq j \middle| X_t \geq \frac{n}{16}\right] \leq \frac{2\lambda^2}{2^j n} \exp\left(-\frac{f_{\max}(t)\lambda}{n}\right) \leq \frac{r}{(1+\delta)^j}.$$

Defining the interval bounds $a := \frac{n}{8}$ and $b := \frac{n}{4}$ and

$$\varepsilon := 2r\frac{(2+\delta)}{\delta \ln(1+\frac{\delta}{2})} = \frac{192\lambda}{ne\ln(\frac{3}{2})},$$

we have

$$\tau := \frac{(b-a)}{2\varepsilon} = \frac{\frac{n}{8}ne\ln\frac{3}{2}}{2\cdot 192\lambda} = \frac{e\ln\frac{3}{2}}{3072}\cdot\frac{n^2}{\lambda} = \Theta\left(\frac{n^2}{\lambda}\right).$$

Therefore, by Theorem 31 we obtain that the probability that X_t exceeds $\frac{n}{4}$ in less than τ iterations if $X_{t_0} < \frac{n}{8}$ is

$$\begin{aligned}\max\left\{4, \frac{2\delta}{r(2+\delta)}\right\} \cdot \frac{(b-a)}{2\varepsilon}\left(1+\frac{\delta}{2}\right)^{-\frac{b-a}{2}} &\leq \frac{2n}{\lambda}\cdot\frac{e\ln\frac{3}{2}}{3072}\cdot\frac{n^2}{\lambda}\cdot\left(\frac{3}{2}\right)^{-\frac{n}{8}} \leq \frac{n^3}{\lambda^2}\left(\frac{3}{2}\right)^{-\frac{n}{8}} \\ &= e^{-\Omega(n)}.\end{aligned}$$

Finally, we conclude that the probability that X_t exceeds $\frac{n}{4}$ in τ iterations is at most

$$\Pr\left[X_{t_0} \geq \frac{n}{8}\right] + \Pr\left[X_{t_0} < \frac{n}{8}\right] e^{-\Omega(n)} = e^{-\Omega(n)} + e^{-\Omega(n)} = e^{-\Omega(n)}.$$

□

An Upper Bound for the Progress in One Iteration

With the additional assumption that the current fitness is not too low, argued for in the previous subsection, we now show that $f_{\max}(t)$ does not increase by more than $O(\frac{\lambda}{n})$ in expectation per iteration. Since we shall later need not a bound on the expected progress, but on the distribution, we formulate our result in this language.

Lemma 77. *Let $\lambda \leq \frac{n}{2}$. If $f_{\max}(t) \in [\frac{n}{2}..\frac{3n}{4}]$ and $f(x) \geq \frac{n}{4}$, then the probability that $f_{\max}(t+1) \geq f_{\max}(t)+j$ is at most*

$$(31+3j)\frac{\lambda}{2^j n}$$

for all $j \in \mathbb{N}$ and $t \in \mathbb{N}$.

Proof. Let $i+1$ be the position of the first zero-bit in x, that is $f(x)+1$. To obtain a better individual in one iteration we must obtain x' such that $x'_{i+1} = 1$. We distinguish the two cases of such outcomes of the mutation phase.

Case 1: $x'_{i+1} = 1$ and $f(x') \leq f_{\max}(t)$. This happens when we either have $f(x') < f(x)$, but fortunately flipped the $i+1$-st bit in x', or when we have $f(x') \in [f(x)+1..f_{\max}(t)]$. By Lemma 72, the probability of the first situation is at most $\frac{4\lambda}{n}$. By Lemma 73 the probability of the second situation is at most

$$\frac{\lambda^2}{n} \exp\left(-\frac{f(x)\lambda}{n}\right) \leq \frac{\lambda^2}{n} e^{-\frac{\lambda}{4}} \leq \frac{32\lambda}{en},$$

where the last inequality follows from Lemma 14. Therefore, the probability of this case is at most

$$\frac{4\lambda}{n} + \frac{32\lambda}{en} = \left(4 + \frac{32}{e}\right)\frac{\lambda}{n} =: p_1. \tag{52}$$

To increase $f_{\max}(t)$ by at least $j \in \mathbb{N}$ from such an x', the algorithm must create a crossover offspring from x and x' that satisfies the following conditions.

1) The bits in positions $[1..i]$ that are zeros in x' must be taken from x. Since we do not know the number of such bits, we only argue that the probability of this event is at most one.
2) The $i+1$-st bit must be taken from x'; the probability of this event is $\frac{1}{\lambda}$.
3) The bits of the offspring in positions $[i+2..f_{\max}(t)+1]$ must be ones. Since we do not understand the distribution of these bits, we bound the probability of this event with one.
4) The bits of the offspring in positions $[f_{\max}(t)+2..f_{\max}(t)+j]$ must be ones. The probability of this event is $\frac{1}{2^{j-1}}$, since these bits independently are equally likely to be one or zero both in x and x'.

Taking all four events together, we see that for all $m \in [1..\lambda]$ and for all $j \in \mathbb{N}$ we have

$$\Pr[f(y^{(m)}) \geq f_{\max}(t) + j \mid f(x') \leq f_{\max}(t) \text{ and } x'_{i+1} = 1] \leq \frac{1}{2^{j-1}\lambda}.$$

By the union bound and by Eq. (52) we conclude that the probability $p_{C_1}(j)$ that we create x' of fitness not greater than $f_{\max}(t)$ and then generate y with fitness at least $f_{\max}(t)+j$ is at most

$$p_{C_1}(j) \leq p_1 \sum_{m=1}^{\lambda} \frac{1}{2^{j-1}\lambda} \leq \left(8 + \frac{64}{e}\right)\frac{\lambda}{2^j n}.$$

Case 2: $f(x') > f_{\max}(t)$. In this case we increase $f_{\max}(t)$ by at least j if $f(x') \geq f_{\max}(t)+j$ or, for some $k \in [1..j-1]$, we have $f(x') = f_{\max}(t)+k$ and from this x' we obtain y such that $f(y) \geq f(x') + (j-k)$. So the probability $p_{C_2}(j)$ that any of these events occurs is

$$\begin{aligned} p_{C_2}(j) &= \Pr[f(x') \geq f_{\max}(t)+j] + \sum_{k=1}^{j-1} q_k \Pr[f(x') = f_{\max}(t)+k] \\ &\leq \Pr[f(x') \geq f_{\max}(t)+j] + \sum_{k=1}^{j-1} q_k \Pr[f(x') \geq f_{\max}(t)+k], \end{aligned} \tag{53}$$

where we define $q_k := \Pr[f(y) \geq f(x') + (j-k) \mid f(x') \geq f_{\max}(t)+k]$ for brevity.

To have $f(y^{(m)}) \geq f(x') + (j-k)$ for some particular offspring $y^{(m)}$ we need the crossover to take the $i+1$-st bit from x' (the probability of this is $\frac{1}{\lambda}$) and we need the bits in positions $[f(x')+1..f(x')+(j-k)-1]$ to be ones. Since these bits are uniformly random both in x and in x' (except bit $f(x')+1$, which is zero in x' and must be taken from x with probability $\frac{\lambda-1}{\lambda}$), the probability of this is $\frac{1}{2^{j-k-1}} \cdot \frac{\lambda-1}{\lambda} \leq \frac{1}{2^{j-k-1}}$. By the union bound over all offspring of the crossover phase we obtain

$$q_k \leq \sum_{m=1}^{\lambda} \frac{1}{\lambda 2^{j-k-1}} = \frac{1}{2^{j-k-1}}.$$

Recall that by Lemma 74 for all $k \in \mathbb{N}$ we have

$$\begin{aligned}\Pr[f(x') \geq f_{\max}(t)+k] &\leq \frac{\lambda^2}{2^{k-1}n} \exp\left(-\frac{f_{\max}(t)\lambda}{n}\right) \\ &\leq \frac{\lambda^2}{2^{k-1}n} e^{-\frac{\lambda}{2}} \leq \frac{4\lambda}{e2^k n},\end{aligned}$$

where the last inequality follows from Lemma 14.

Substituting the probabilities in Eq. (53) with these estimates we obtain

$$\begin{aligned}p_{C_2}(j) &\leq \frac{4\lambda}{e2^j n} + \sum_{k=1}^{j-1} \frac{4\lambda}{e2^k n} \cdot \frac{1}{2^{j-k-1}} \\ &= \frac{4\lambda}{en}\left(\frac{1}{2^j} + \frac{j-1}{2^{j-1}}\right) = \frac{4\lambda(2j-1)}{e2^j n}.\end{aligned}$$

We conclude that the probability that after the crossover phase we obtain an individual with fitness by at least j greater than $f_{\max}(t)$ is at most

$$\begin{aligned}p_{C_1}(j) + p_{C_2}(j) &\leq \left(8 + \frac{64}{e}\right)\frac{\lambda}{2^j n} + \frac{4}{e}(2j-1)\frac{\lambda}{2^j n} \\ &\leq (31+3j)\frac{\lambda}{2^j n}.\end{aligned}$$

□

To shorten the further proofs we also give the following bound on the progress in one iteration, which does not depend on the current value of $f_{\max}(t)$.

Lemma 78. *If $\lambda \leq \frac{n}{2}$, the probability that $f_{\max}(t)$ increases by more than $4\log_2(n)$ in one iteration is at most $\frac{3}{2n}$.*

Proof. By Corollary 3 the probability that the fitness of the mutation winner $f(x')$ is greater than $f_{\max}(t)$ by more than $2\log_2 n$ is at most $\frac{1}{2n}$.

Notice that each bit in positions $[\max\{f_{\max}(t), f(x')\}+2..n]$ has equal chances to be one or zero both in the current individual x and in the mutation winner x', which also applies to any offspring created in the crossover phase. Therefore, conditional on $f(x') \leq f_{\max}(t) + 2\log_2(t)$, to obtain an offspring $y^{(m)}$ with fitness at least $f_{\max}(t) + 4\log_2 n$ we need the bits in at least

$2\log_2 n - 1$ positions (namely, in positions $[\max\{f_{\max}(t), f(x')\} + 2..f_{\max}(t) + 4\log_2 n]$) to be ones in this offspring. The probability of this is at most $\frac{1}{2^{2\log_2 n - 1}} \leq \frac{2}{n^2}$. By the union bound the probability that it happens in at least one offspring in the crossover phase is therefore at most $\frac{2\lambda}{n^2} \leq \frac{1}{n}$.

Hence, the progress in $f_{\max}(t)$ is greater than $4\log_2 n$ with probability at most $\frac{1}{2n} + (1 - \frac{1}{2n})\frac{1}{n} \leq \frac{3}{2n}$. □

The Final Drift Argument

With the preparations already made in this subsection, we are now ready to prove the lower bound of our main result. We argue that $f_{\max}(t)$ has a drift of only $O(\frac{\lambda}{n})$ in the interval $[\frac{n}{2}, \frac{3n}{4}]$. We show that $f_{\max}(t)$ is close to $\frac{n}{2}$ when $f_{\max}(t)$ first enters this interval. Hence $f_{\max}(t)$ has to increase by $\Omega(n)$ to reach or exceed $\frac{3n}{4}$. With a drift of only $O(\frac{\lambda}{n})$, this takes expected time of at least $\Omega(\frac{n^2}{\lambda})$ iterations by the additive drift theorem.

Proof of Theorem 35. We first analyze the state of the algorithm in the first iteration t_0 such that $f_{\max}(t) \geq \frac{n}{2}$. If $t_0 \geq \tau$ (where τ is the threshold from Lemma 76), then the runtime is already $\Omega(\frac{n^2}{\lambda})$ iterations. Otherwise, by Lemma 76 at this iteration we have $f(x) \geq f_{\max}(t_0) - \frac{n}{4} \geq \frac{n}{4}$ with probability $1 - e^{-\Omega(n)}$.

Restricting ourselves to the case $f(x) \geq \frac{n}{4}$ we note that the probability that $f_{\max}(t_0) > \frac{n}{2} + 4\log_2(n)$ is at most the probability that $f_{\max}(t)$ is increased by more than $4\log_2(n)$ in one iteration. By Lemma 78 we have

$$\begin{aligned}\Pr[f_{\max}(t_0) > \tfrac{n}{2} + 4\log_2(n)] &\leq \Pr[f_{\max}(t_0) \geq f_{\max}(t_0 - 1) + 4\log_2(n)] \\ &\leq \frac{3}{2n}.\end{aligned}$$

Let T be the first time when $f_{\max}(t) > \frac{3n}{4}$. Then by Lemma 77 for all $t \in [t_0, T-1]$ we have

$$\begin{aligned}\Delta_{f_{\max}} &= E[f_{\max}(t+1) - f_{\max}(t)] \\ &\leq \sum_{j=1}^{+\infty} \Pr[f_{\max}(t+1) \geq f_{\max}(t) + j] \\ &\leq \sum_{j=1}^{+\infty} \frac{(31+3j)\lambda}{n2^j} = \frac{37\lambda}{n}.\end{aligned}$$

Finally, applying the additive drift theorem we obtain

$$E[T] \geq \frac{\frac{3n}{4} - f_{\max}(t_0)}{\Delta_{f_{\max}}} \geq \frac{\frac{n}{4} - 4\log_2(n)}{\frac{37\lambda}{n}} = \Omega\left(\frac{n^2}{\lambda}\right).$$

This bound holds if $t_0 < \tau$ and $f_{\max}(t) \leq \frac{n}{2} + 4\log_2(n)$. The probability that both of these conditions are satisfied is at least $1 - \frac{3}{2n} - e^{-\Omega(n)} \geq 1 - \frac{2}{n}$, if n is large enough. Therefore, the expected runtime of the algorithm is at least $(1 - \frac{2}{n})\Omega(\frac{n^2}{\lambda}) = \Omega(\frac{n^2}{\lambda})$ iterations.

□

4.3.3 Lower Bound for Adaptive λ

In this section we prove that any strategy of adjusting the value of λ during the algorithm run cannot speed up the algorithm (in terms of the asymptotic number of fitness evaluations). We consider a modification of the $(1+(\lambda,\lambda))$ GA such that it chooses the value of λ from $[1..\lfloor\frac{n}{2}\rfloor]$ before each iteration (probably not from the power-law distribution). We call this algorithm an *adaptive* $(1+(\lambda,\lambda))$ *GA*.

Theorem 36. *The adaptive* $(1+(\lambda,\lambda))$ *GA with any strategy of choosing* λ *needs* $\Omega(n^2)$ *fitness evaluations to find the optimum of* LEADINGONES.

The proof of this theorem builds on the observation that while $f_{\max}(t) \in [\frac{n}{2}..\frac{3n}{4}]$ the expected progress in one iteration is $O(\frac{\lambda}{n})$. Since the algorithm performs exactly 2λ fitness evaluations in each iteration, the expected progress per one call of the fitness function is therefore $O(\frac{1}{n})$. This intuitive observation, however, cannot be directly transformed into a rigorous proof, since the fitness evaluations come into groups of size 2λ and progress can occur only after such group. For this reason the standard drift arguments are not applicable here.

Lemma 79. *Consider a run of the adaptive* $(1+(\lambda,\lambda))$ *GA. Let* $n \geq 4$ *and let* t_0 *be the first iteration which starts with* $f_{\max}(t) \geq \frac{n}{2}$*. If* $f_{\max}(t_0) \leq \frac{n}{2} + 4\log_2(n)$ *and* $f(x) \geq \frac{n}{4}$ *in the start of iteration* t_0*, then the runtime until* $f_{\max}(t) \geq \frac{3n}{4}$ *is* $\Omega(n^2)$ *fitness evaluations with probability* $1 - e^{-\Omega(n)}$.

Proof. Let λ_t be the value of λ chosen in the start of iteration t. Let $X_t := \sum_{i=t_0}^{t} 2\lambda_i$, which is the number of fitness evaluation the algorithm performed since iteration t_0 until iteration t (inclusive). Consider the random variable $Y_t := f_{\max}(t) - \alpha\frac{X_t}{n}$ with constant $\alpha := \frac{270}{\ln(3/2)}$. As long as $Y_t < \frac{5n}{8}$ and $X_t \leq \frac{n^2\ln(3/2)}{2160}$, we have

$$f_{\max}(t) = Y_t + \alpha\frac{X_t}{n} < \frac{5n}{8} + \frac{n}{8} \leq \frac{3n}{4}.$$

Now we aim to show that Y_t does not reach $\frac{5n}{8}$ in n^2 iterations with high probability, which means that with at most the same probability $f_{\max}(t)$ reaches $\frac{3n}{4}$ only after $\frac{n^2\ln(3/2)}{2160} = \Omega(n^2)$ fitness evaluations. To use Theorem 1 we compute the expected drift $\Delta_Y(\gamma)$ of Y_t.

$$\begin{aligned}
\Delta_Y(\gamma) :=& E[\exp(\gamma(Y_{t+1} - Y_t)) \mid Y_0, \dots, Y_t] \\
=& E\left[\exp\left(\gamma\left(f_{\max}(t+1) - f_{\max}(t) - \alpha\frac{\lambda_t}{n}\right)\right) \,\middle|\, Y_t\right] \\
\leq& \exp\left(-\gamma\alpha\frac{\lambda_t}{n}\right)\left(\Pr[f_{\max}(t+1) = f_{\max}(t)]\right. \\
&\left. + \sum_{j=1}^{+\infty}\Pr[f_{\max}(t+1) = f_{\max}(t)+j]e^{\gamma j}\right).
\end{aligned}$$

By Lemma 77 this is at most

$$\Delta_Y(\gamma) \leq \exp\left(-\gamma\alpha\frac{\lambda_t}{n}\right)\left(1+\sum_{j=1}^{+\infty}\frac{(31+3j)\lambda_t}{2^j n}e^{\gamma j}\right).$$

With $\gamma := \ln\frac{3}{2}$ and Lemma 16 we obtain

$$\begin{aligned}\Delta_Y(\gamma) &\leq \exp\left(-\frac{270\lambda_t}{n}\right)\left(1+135\frac{\lambda_t}{n}\right)\\ &\leq \max\left\{1-\frac{1}{4}, 1-\frac{135\lambda_t}{2n}\right\} \leq 1-\frac{1}{n},\end{aligned}$$

if $n \geq 4$. Hence, we define $p := n$.

Let T be the first iteration when $Y_t \geq \frac{5n}{8}$. By the negative drift theorem (Theorem 1) we have

$$\begin{aligned}\Pr[T \leq t_0 + n^2] &\leq n^2 p \exp\left(-\gamma\left(\frac{5n}{8} - Y_{t_0}\right)\right)\\ &\leq n^3 \exp\left(-\ln\frac{3}{2}\left(\frac{n}{8} - 2\log_2 n\right)\right) = e^{-\Omega(n)}.\end{aligned}$$

□

Now to prove Theorem 36 it is enough to prove that the initial conditions of Lemma 79 are satisfied with high probability.

Lemma 80. *Let t_0 be the first iteration when $f_{\max}(t) \geq \frac{n}{2}$. There exists some $\tau = \Theta(n^2)$ such that either $t_0 \geq \tau$ or $t_0 < \tau$ and the conditions of Lemma 79 are satisfied with probability $1 - e^{-\Omega(n)}$.*

Proof. We aim to find τ such that $t_0 < \tau$ implies

(i) $f(x) \geq \frac{n}{4}$ at iteration t_0 and

(ii) $f_{\max}(t) \leq \frac{n}{2} + 4\log_2(n)$.

To prove that the first condition holds we argue that Lemma 76 also holds for some $\tau = \Theta(n^2)$. Not to repeat the proof, we just notice that to obtain this result we need to change the definition of r in the proof of Lemma 76 and take $r := \frac{2048}{e^2 n}$, which by Lemma 15 still satisfies the further arguments (independently on the choice of λ in each iteration), with only exception that we also redefine

$$\varepsilon := \frac{12288}{ne^2\ln\frac{3}{2}}$$

and

$$\tau := \frac{e^2\ln\frac{3}{2}n^2}{196608} = \Theta(n^2).$$

To prove that the second condition holds we notice that Lemma 78 holds for any λ_t, therefore, in the iteration t_0-1 the value of $f_{\max}(t)$ increases by more than $4\log_2 n$ with probability at most $\frac{3}{2n}$.

Finally, by the union bound we have that the probability that both conditions hold at iteration τ (as we redefined it) is at least $1 - e^{-\Omega(n)} - \frac{3}{2n}$. □

Theorem 36 follows directly from Lemmas 79 and 80.

4.3.4 Upper Bound

We denote by i the current fitness, that is, $f(x)$. The main idea of our proof is to show that the mutation phase winner x' has the $i+1$-st bit equal to one with probability $\Omega(\frac{\lambda}{n})$. Afterwards, we prove that conditional on this, there is at least a constant probability to generate an offspring y such that the i-th bit of y is taken from x' and all other bits that differ in x and x' are taken from x. For the latter estimate to hold we assume that ℓ is between $\frac{\lambda}{2}$ and 2λ. Combining the estimates we get an upper bound on the expected runtime of $O(\frac{n^2}{\lambda})$ iterations. These arguments bring us the following theorem, which is the main result of this section.

Theorem 37. *The expected runtime of the $(1 + (\lambda, \lambda))$ GA on the LEADINGONES problem is $O(\frac{n^2}{\lambda})$ iterations or $O(n^2)$ fitness evaluations.*

To avoid some technical difficulties and simplify the rest of this section we first prove this bound for relatively trivial case when $\lambda < 4$.

Theorem 38. *If $\lambda < 4$, the expected runtime of the $(1 + (\lambda, \lambda))$ GA on the LEADINGONES problem is $O(n^2)$ iterations.*

Proof. The line of the proof generally repeats the one for the $(1 + 1)$ EA. In order to obtain progress in one iteration, it suffices to satisfy the following conditions.

1) At least one mutant is better than the current individual (so that the mutation winner is better than the current individual as well).
2) At least one crossover offspring took the critical bit from the mutation winner (since all previous bits are ones in both parents, this results into crossover winner which is better than the current individual).

The probability p_1 that some mutant is better than the current individual x is the probability that the mutation does not flip any bit in the prefix and it flips the critical bit. We estimate it as

$$p_1 \geq \left(1 - \frac{\lambda}{n}\right)^n \frac{\lambda}{n} \geq \left(\frac{1}{4}\right)^\lambda \frac{\lambda}{n} \geq \left(\frac{1}{4}\right)^3 \frac{3}{n} \geq \frac{3}{64n}.$$

The probability p_2 that at least one mutant is better than the parent is not less than p_1. The probability p_3 that in the crossover phase we take the critical bit from the winner of the mutation phase in one particular offspring is $\frac{1}{\lambda} \geq \frac{1}{3}$. The probability p_4 that we do it in at least one offspring is at least $p_3 \geq \frac{1}{3}$. Finally, we have that the probability to create a better offspring in one iteration is at least $p_2 p_4 \geq \frac{1}{64n}$. Since we need at most n improvements to reach the optimum the total runtime is at most $64n \cdot n = O(n^2)$ iterations. □

Further in this section we assume that λ is at least 4.

Mutation Phase

In this section we estimate the probability to obtain an individual with the $i+1$-st bit flipped as the winner of the mutation phase. First we argue that with at least constant probability the number of the flipped bits is around λ.

Lemma 81. *If $\lambda \geq 4$, the number ℓ of the bitts flipped by the mutation operator is in $[\frac{\lambda}{2}, 2\lambda]$ with probability at least $1 - e^{-\frac{\lambda}{3}} - e^{-\frac{\lambda}{4}} \geq \frac{1}{9}$.*

Proof. Recall that the number of bits to flip ℓ is chosen according to the binomial distribution $\mathrm{Bin}(n, \frac{\lambda}{n})$. This implies that its expectation $E[\ell] = n\frac{\lambda}{n} = \lambda$. By Chernoff bounds we have

$$\Pr[\ell \geq 2E[\ell]] \leq \exp\left(-\frac{\lambda}{3}\right),$$

and

$$\Pr\left[\ell \leq \frac{E[\ell]}{2}\right] \leq \exp\left(-\frac{\lambda}{8}\right).$$

Therefore, the probability that $\ell \in [\frac{\lambda}{2}, 2\lambda]$ is at least

$$\Pr\left[\ell \in \left[\frac{\lambda}{2}, 2\lambda\right]\right] \geq 1 - \exp\left(-\frac{\lambda}{3}\right) - \exp\left(-\frac{\lambda}{8}\right),$$

which is at least a positive constant (approximately, $0.1298 \geq \frac{1}{9}$) for all $\lambda \geq 4$. □

Conditional on that the number of the flipped bits does not differ much from λ, we further condition on the fitness j of the winner of the mutation phase x' and consider two possible outcomes.

Lemma 82. *If $\ell \geq \frac{\lambda}{2}$ and the fitness j of the winner of the mutation phase x' is less than the current fitness i, then the probability that the $i+1$-st bit is one in x' is at least $\frac{\lambda}{4n}$.*

Proof. Since the fitness of x' is j, the $j+1$-st bit was flipped by the mutation operator and all other $\ell - 1$ flipped bits are to the right from the $j+1$-st bit. The positions of these flipped bits were chosen uniformly at random and were not affected by the selection. Therefore, for each bit to the right of the $j+1$-st bit (including the $i+1$-st bit) the probability to be flipped is

$$\frac{\ell - 1}{n - j - 1} \geq \frac{\ell - 1}{n} \geq \frac{\frac{\lambda}{2} - 1}{n} \geq \frac{\lambda}{4n}.$$

□

Lemma 83. *If $\ell \geq \frac{\lambda}{2}$ and the fitness j of the winner of the mutation phase x' is not less than the current fitness i, then the probability that the $i+1$-st bit is one in x' is at least $\frac{\lambda}{2n}$.*

Proof. From the conditions of the lemma we have that there is at least one mutant with fitness at least i. Assume that there are k such mutants with $k \geq 1$. For each of them the positions of the ℓ flipped bits are uniformly distributed in $[i+1..n]$. Hence, the probability that the $i+1$-st bit is flipped in one such mutant is $\frac{\ell}{n-i}$.

To have the $i+1$-st bit equal to zero in the mutation winner x', we need to have it equal to zero in all k mutants with fitness at least i, otherwise there is a mutant with fitness at least $i+1$ that is selected as x'. Since the mutants are generated independently, the probability this event does not occur is at least

$$1 - \left(1 - \frac{\ell}{n-i}\right)^k \geq \frac{\ell}{n-i} \geq \frac{\ell}{n} \geq \frac{\lambda}{2n}.$$

□

From Lemmas 81, 82 and 83 we conclude the main result of this subsection.

Lemma 84. *If $\lambda \geq 4$, with probability at least $\frac{\lambda}{36n}$ the winner of the mutation phase x' is such that*
- $x'_{i+1} = 1$ *and*
- $H(x,x') \leq 2\lambda$, *where $H(x,x')$ is Hamming distance between x' and the current individual x.*

Proof. By the definition of conditional probability we have

$$\begin{aligned}\Pr[x'_{i+1} = 1 \text{ and } H(x,x') \leq 2\lambda] &\geq \Pr\left[x'_{i+1} = 1 \text{ and } \ell \in \left[\frac{\lambda}{2}, 2\lambda\right]\right] \\ &= \Pr\left[x'_{i+1} = 1 \middle| \ell \in \left[\frac{\lambda}{2}, 2\lambda\right]\right] \\ &\quad \cdot \Pr\left[\ell \in \left[\frac{\lambda}{2}, 2\lambda\right]\right].\end{aligned}$$

By Lemma 81 we have

$$\Pr\left[\ell \in \left[\frac{\lambda}{2}, 2\lambda\right]\right] \geq \frac{1}{9},$$

and by Lemmas 82 and 83 we have

$$\Pr\left[x'_{i+1} = 1 \middle| \ell \in \left[\frac{\lambda}{2}, 2\lambda\right]\right] \geq \frac{\lambda}{4n}.$$

Finally, we obtain

$$\Pr[x'_{i+1} = 1 \text{ and } H(x,x') \leq 2\lambda] \geq \frac{\lambda}{4n} \cdot \frac{1}{9} = \frac{\lambda}{36n}.$$

□

Crossover Phase

Now we consider the probability to have progress in the crossover phase. For that purpose we estimate the probability to get y such that y_{i+1} is a one-bit and none of other bits are changed compared to x (and hence we increase the fitness by at least 1). We estimate the probability to obtain progress in one particular offspring in the following lemma.

Lemma 85. *Let $\lambda \geq 4$. Assume that $x'_{i+1} = 1$ and the Hamming distance between x and x' is at most 2λ. Then for any offspring $y^{(m)}$ we have $\Pr[f(y^{(m)}) > f(x)]$ is at least $\frac{1}{8\lambda}$.*

Proof. To obtain an offspring $y^{(m)}$ that is better than x we can take bit $i+1$ from x' with probability $\frac{1}{\lambda}$ and take all other $2\lambda - 1$ bits that differ in x and x' from x with probability $(1 - \frac{1}{\lambda})$ each. Therefore, we have

$$\Pr[f(y^{(m)}) > f(x) \mid x'_{i+1} = 1 \cap H(x,x') \leq 2\lambda] \geq \frac{1}{\lambda}\left(1 - \frac{1}{\lambda}\right)^{2\lambda-1}$$

$$\geq \frac{1}{8\lambda}.$$

□

The winner of the crossover phase y is strictly better than x, if at least one offspring $y^{(m)}$ is better than x. Hence, we have

$$\begin{aligned}\Pr[f(y) > f(x) \mid x'_{i+1} = 1 \cap H(x,x') \leq 2\lambda] &\geq 1 - \left(1 - \frac{1}{8\lambda}\right)^{\lambda} \\ &\geq 1 - e^{-\frac{1}{8}}.\end{aligned}$$

Now we have all arguments to prove Theorem 37.

Proof of Theorem 37. The probability that the algorithm improves its current fitness in one iteration is at least the probability that the three consequent events happen.

- The number of flipped bits ℓ is in $[\frac{\lambda}{2}, 2\lambda]$.
- The $i+1$-st bit is flipped in x'.
- There is an offspring $y^{(m)}$ created in the crossover phase such that it differs from x only in the $i+1$-st bit.

By Lemmas 84 and 85 the probability of this sequence is at least

$$\frac{\lambda}{36n}\left(1 - e^{-\frac{1}{8}}\right) = \Omega\left(\frac{\lambda}{n}\right).$$

Therefore, the $(1+(\lambda,\lambda))$ GA spends expected number of $O(\frac{n}{\lambda})$ iterations before it improves the fitness. Since there can be at most n improvements of the fitness before the optimum is found, the total runtime of the $(1+(\lambda,\lambda))$ GA on the LeadingOnes is $O(\frac{n^2}{\lambda})$. Since exactly 2λ fitness evaluations are made in each iteration, the runtime is $O(n^2)$ fitness evaluations. □

4.3.5 Upper Bound for Adaptive λ

In the same manner as in Section 4.3.3 we observe that the average expected progress per fitness evaluation is $\Omega(\frac{1}{n})$. From this we can also conclude that any adaptive choice of λ does not speed up or slow down the runtime by more than a constant factor. However, we cannot use the drift argument out of the box, since the progress comes only after an iteration which consists of 2λ fitness evaluations. We overcome this with a similar trick as in Section 4.3.3 and prove the following theorem.

Theorem 39. *The adaptive $(1+(\lambda,\lambda))$ GA with any strategy of choosing the value of λ finds an optimum of LeadingOnes in $O(n^2)$ fitness evaluations with probability $1 - e^{-n+o(n)}$.*

From the previous section, we derive the following lower bound on the probability for a fitness increase.

Lemma 86. *If the adaptive $(1+(\lambda,\lambda))$ GA chooses λ_t in the start of iteration t, then the probability that $f(x)$ increases in this iteration is at least $\frac{\lambda_t}{324n}$.*

Proof. If the algorithm chooses $\lambda_t < 4$, then with the same arguments as in Theorem 38 we can show that the probability of progress in one iteration is at least $(\frac{1}{4})^{\lambda_t}\frac{1}{n} \geq \frac{\lambda_t}{192n} > \frac{\lambda_t}{324n}$.

Otherwise, by Lemma 84 and Lemma 85 we have that the probability of progress is at least

$$\frac{\lambda_t}{36n}\left(1-e^{-\frac{1}{8}}\right) > \frac{\lambda_t}{324n}.$$

□

Now, using the similar ideas as in Lemma 79 we prove the main result of this section.

Proof of Theorem 39. For all $t \in \mathbb{N}$, let λ_t be the value of λ chosen at the start of iteration t and x_t be the current individual in the start of iteration t. Let $X_t := \sum_{i=0}^{t} 2\lambda_i$, which is the number of fitness evaluations the algorithm performed until iteration t (inclusive). Consider the random variable

$$Y_t = \frac{X_t}{1296en} - f(x_t).$$

If $Y_t \leq n$ and $X_t \geq 2592en^2 = O(n^2)$, then $f(x_t)$ must be at least n. If Y_t does not exceed n before X_t reaches $2592en^2$, the optimum is found by the moment X_t exceeds $2592en^2$. We show that Y_t does not exceed n for at least $1296en^2$ iterations, which is surely enough for X_t to reach $2592en^2$ even if the algorithm always chooses $\lambda_t = 1$.

To invoke Theorem 1, we compute $E[e^{\gamma(Y_{t+1}-Y_t)}]$ with $\gamma := 1$. By Lemma 86 we have

$$\begin{aligned} E\left[e^{Y_{t+1}-Y_t}\right] &= E\left[\exp\left(\frac{2\lambda_t}{1296en} - f(x_{t+1}) + f(x_t)\right)\right] \\ &\leq \exp\left(\frac{\lambda_t}{648en}\right)\left(\frac{\lambda_t}{324n}e^{-1} + \left(1-\frac{\lambda_t}{324n}\right)\right) \\ &= \exp\left(\frac{\lambda_t}{648en}\right)\left(1-\frac{(e-1)\lambda_t}{324en}\right). \end{aligned}$$

Since for all $\lambda_t \in [1..\frac{n}{2}]$ we have $\frac{\lambda_t}{648en} \in [0,1]$, by Lemma 17 we also have

$$E\left[e^{Y_{t+1}-Y_t}\right] \leq 1 - \frac{(e-2)\lambda_t}{648en}$$

Let T be the first time when Y_t exceeds n. By the negative drift theorem (Theorem 1) we have

$$\Pr[T \leq 1296en^2] \leq 1296en^2 \cdot \frac{648en}{(e-2)\lambda_t} \cdot e^{-n} = e^{-n+o(n)},$$

which is an upper bound on the probability that algorithm does not find the optimum in $1296en^2$ iterations. □

4.4 The Runtime Analysis of the $(1+(\lambda,\lambda))$ GA with on JUMP

Clearly the usual application of evolutionary algorithms are problems with multimodal landscapes, that is, with non-trivial local optima, and these local optima usually present a difficulty for evolutionary algorithms. In the runtime analysis perspective multimodal problems

have displayed very different optimization behaviors. For example, on multimodal landscapes it has been observed that crossover can recombine solutions into significantly better ones [90], that mutation rates significantly larger than $\frac{1}{n}$ can be preferable [57], and that estimation-of-distribution algorithms can significantly outperform classic algorithms [79, 23].

In this light and given that all previous runtime analyses for the $(1+(\lambda,\lambda))$ GA consider unimodal problems or problems that are sufficiently close to unimodal for the multimodality to have no effect, we feel that it is the right time to now investigate how the $(1+(\lambda,\lambda))$ GA optimizes multimodal problems. Being the prime multimodal benchmark in runtime analysis, we regard jump functions.

The main result of this section is a runtime analysis of the $(1+(\lambda,\lambda))$ GA with static parameters on jump functions for all jump sizes $k \in [2..\frac{n}{16}]$. We deviate from the standard parameters and consider arbitrary values for the mutation rate p, the crossover bias c, and the offspring population size λ. We motivate it by the observation that the standard parameters setting aim at making small steps (but with high success probability).More precisely, the distance between the current individual x and any offspring $y^{(i)}$ of the crossover phase follows a binomial distribution $\mathrm{Bin}(n,\frac{1}{n})$ which, as we know from [57], is not suitable for making jumps of size k. We also allowed different offspring population sizes λ_m and λ_c for the mutation and crossover phase, which however did not lead to stronger runtime guarantees.

For all $k \in [2..\frac{n}{16}]$ and for arbitrary values of these four parameters except for the only constraint $p \geq \frac{2k}{n}$, we prove that the $(1+(\lambda,\lambda))$ GA crosses the fitness valley in expected time (number of fitness evaluations) at most

$$E[T] \leq \frac{4(\lambda_m+\lambda_c)}{q_\ell \min\{1,\lambda_m(\frac{p}{2})^k\}\min\{1,\lambda_c c^k(1-c)^{2pn-k}\}},$$

where q_ℓ is a constant from $[0.1,1]$, if it starts in the local optimum of $\textsc{Jump}_k$. Ignoring the hidden constants in the resulting $e^{O(k)}$ factor, this bound is optimized for $p = c = \sqrt{\frac{k}{n}}$ and $\lambda_m = \lambda_c = n^{k/2}k^{-k/2}$ and then gives

$$E[T] \leq n^{k/2}e^{O(k)}k^{-k/2}.$$

When not starting in the local optimum, but with an arbitrary initial solution (or the usual random initialization), the $(1+(\lambda,\lambda))$ GA reaches the local optimum in an expected time of $ne^{O(k)}$ iterations, if $p = c = \sqrt{\frac{k}{n}}$ and λ_m and λ_c are at least $\frac{n}{k}$. With slightly smaller values for the population sizes, namely, $\lambda_m = \lambda_c = n^{(k-1)/2}k^{-k/2}$ this gives us the total expected runtime of

$$E[T] \leq n^{(k+1)/2}e^{O(k)}k^{-k/2}.$$

As for the previous results on $\textsc{Jump}_k$, a speed-up over classic algorithms is also observed for larger ranges of parameters, though these are harder to describe in a compact fashion (see Corollary 5 for the details).

The result above shows that the power of the $(1+(\lambda,\lambda))$ GA becomes much more visible for jump functions than for the problems regarded in previous works. Concerning the optimal parameter values, we observe that they differ from those that were optimal in the previous works. In particular, the relation of mutation rate and crossover bias is different. Whereas in previous works $pcn = 1$ was a good choice, we now have $pcn = k$. A moment's thought, however, shows that this is quite natural, or, being more cautious, at least fits to the previous results. We recall

that *pcn* is the expected Hamming distance of the parent to an individual generated from one isolated application of mutation and crossover. The previous works suggested that this number should be one, since one is also the expected distance of an offspring generated the classic way, that is, via standard bit mutation with mutation rate $\frac{1}{n}$.

Now for the optimization of jump functions, where a non-trivial local optimum has to be left, it makes sense to put more weight on larger moves in the search space. More specifically, the work [57] has shown that the optimal mutation rate for the $(1 + 1)$ EA optimizing jump functions is $\frac{k}{n}$. Hence for the classic $(1 + 1)$ EA, the best way of generating offspring is such that they have an expected Hamming distance of k from the parent. Clearly, this remains an intuitive argument, but it shows that also when optimizing multimodal problems, the intuitive approach of previous works, which might help an algorithm designer, gave the right intuition.

Our recommendation when using the $(1 + (\lambda, \lambda))$ GA for multimodal optimization problems would therefore be to choose p and c larger than in previous works, and more speficially, in a way that *pcn* is equal to an estimate for the number of bits the algorithm typically should flip. Here "typically" does not mean that there are actually many moves of this size, but that this is the number of bits the algorithm has to flip most often. For example, when the $(1 + 1)$ EA optimizes a jump function, it will maybe only once move to a search point in distance k, however, it will nevertheless need many offspring in distance k until it finds the right move of this distance.

From our rigorous analysis, we conclude that the $(1 + (\lambda, \lambda))$ GA is even better suited for the optimization of multimodal objective functions, and we hope that the just sketched intuitive considerations help algorithm designers to successfully apply this algorithm to their problems.

4.4.1 Upper Bounds

In this section we analyse the $(1 + (\lambda, \lambda))$ GA with general parameters on JUMP_k and show upper bounds on its runtime. We recede from the standard parameter setting of the $(1 + (\lambda, \lambda))$ GA, since the intuition behind these parameters values (that is, the intent to have only a single bit flipped if we consequently apply mutation and crossover operators) suggests that they are not efficient to escape local optima.

We split our analysis into two parts. First we find the expected time the $(1 + (\lambda, \lambda))$ GA needs to perform a jump to the global optimum when it is already in the local optimum. Then we complete the story by considering the runtime until the $(1 + (\lambda, \lambda))$ GA gets to the local optimum starting in a random bit string.

We do not consider the case when $k = 1$, since JUMP_1 coincides with ONEMAX, which is already well-studied in the context of the $(1 + (\lambda, \lambda))$ GA (see [31] for the full picture). We also omit considering too large values of k (namely, $k > \frac{n}{16}$) since they do not give much new insight about the $(1 + (\lambda, \lambda))$ GA, while they require more complicated arguments for our results to hold.

We also constrain ourselves to the case $p \geq \frac{2k}{n}$ so that once we get to the local optimum we have a decent probability to flip at least $2k$ bits. This implies that an individual with k zero-bits flipped will have a better fitness than any other offspring and therefore selected as the winner of the mutation phase x'. Without this assumption an individual with all zero-bits flipped to one might occur in the fitness valley, thus it is not detected as the mutation phase winner. Hence,

the jump to the global optimum becomes more challenging for the algorithm, which makes this parameter setting not really promising to be effective on multimodal functions.

Escaping the Local Optimum

In this section we analyse how the $(1+(\lambda,\lambda))$ GA leaves the local optimum. Although by runtime we understand the time until the optimum is sampled, it is fair to consider the time until x becomes the optimum for at least two reasons. (i) By disregarding the event that the optimum is sampled in the mutation phase, we still get an upper bound on the runtime. (ii) Since the probability to sample the optimum in the mutation phase is small compared to the probability to sample the optimum in the crossover phase, we expect to lose only a little. Due to the elitist selection the only chance to leave the local optimum is to find the global optimum in one iteration. For this it is sufficient that the following two consecutive events happen.

1) The mutation phase winner x' has all k bits which are zero in the current individual x flipped to one.
2) The crossover winner y takes all k bits which are zero in x from x' and all bits which are zero in x' from x.

We first estimate the probability of the first event and then estimate the probability of the second event conditional on the first one.

We call the mutation phase *successful* if all k zero-bits of x are flipped to one in x' (and possibly some one-bits are flipped to zero) and the number ℓ of the flipped bits is at most $2pn$. We estimate the probability p_m of having a successful mutation phase in the following lemma.

Lemma 87. *Let $k \leq \frac{n}{4}$. If $p \geq \frac{2k}{n}$, then we have*

$$p_m \geq \frac{q_\ell}{2} \min\left\{1, \lambda_m \left(\frac{p}{2}\right)^k\right\},$$

where q_ℓ is as defined in Lemma 24, which is $\Theta(1)$.

Proof. If $\ell \geq 2k$ then we flip at least k one-bits in each mutant, hence the fitness of each mutant is at most $n-k$. Therefore, if there is at least one individual with all k zero-bits flipped, then this individual has a greater value of $\textsc{Jump}_k$ than any other individual which does not have all zero-bits flipped. Hence, such an individual is chosen as the mutation winner x'. Therefore, for a successful mutation phase it suffices that the following two events occur (in this order).

- The number of flipped bits ℓ is in $[pn, 2pn]$.
- The k zero-bits of x are among the ℓ chosen bits in at least one of the λ_m offspring. We call such offspring *good* in this proof.

By Lemma 24 the probability of the first event is $q_\ell \geq 0.1$. We condition on this event in the remainder. The probability $q_m(\ell)$ that one particular offspring is good is $\frac{\binom{n-k}{\ell-k}}{\binom{n}{\ell}}$. By the assumption that $p \geq \frac{2k}{n}$ we have

$$q_m(\ell) = \frac{\binom{n-k}{\ell-k}}{\binom{n}{\ell}} = \frac{(n-k)!}{(\ell-k)!(n-\ell-2k)!} \cdot \frac{\ell!(n-\ell)!}{n!}$$

$$= \frac{\ell(\ell-1)\dots(\ell-k+1)}{n(n-1)\dots(n-k+1)} \geq \left(\frac{\ell-k}{n}\right)^k \geq \left(\frac{pn}{2n}\right)^k = \left(\frac{p}{2}\right)^k.$$

The probability that at least one offspring is good is $1-(1-q_m(\ell))^{\lambda_m}$. By Lemma 21, we estimate

$$\begin{aligned} 1-(1-q_m(\ell))^{\lambda_m} &\geq \frac{1}{2}\min\{1, \lambda_m q_m(\ell)\} \\ &\geq \frac{1}{2}\min\left\{1, \lambda_m \left(\frac{p}{2}\right)^k\right\}. \end{aligned}$$

Therefore, we conclude

$$\begin{aligned} p_m &\geq \Pr[\ell \in [pn, 2pn]] \cdot \frac{1}{2}\min\left\{1, \lambda_m \left(\frac{p}{2}\right)^k\right\} \\ &\geq \frac{q_\ell}{2}\min\left\{1, \lambda_m \left(\frac{p}{2}\right)^k\right\}. \end{aligned}$$

□

Now we proceed with the crossover phase. We call the crossover phase *successful* (conditional on a successful mutation phase) if the winner y takes all bits which are zero in x' from x (where they are one) and all k bits which are zero in x from x' (where they are ones). We denote the probability of a successful crossover phase by p_c.

Lemma 88. *Assume that $k \leq \frac{n}{4}$ and the mutation phase was successful. Then*

$$p_c \geq \frac{1}{2}\min\left\{1, \lambda_c c^k (1-c)^{2pn-k}\right\}.$$

Proof. To generate an optimal solution in one application of the crossover operator we need to take k particular bits from x' and $\ell-k$ particular bits from x. The probability q_c to generate such a crossover offspring is

$$q_c = c^k (1-c)^{\ell-k} \geq c^k (1-c)^{2pn-k},$$

since a successful mutation implies that $\ell \leq 2pn$. The probability to generate at least one such offspring is

$$\begin{aligned} p_c &= 1-(1-q_c)^{\lambda_c} \geq 1-\left(1-c^k(1-c)^{2pn-k}\right)^{\lambda_c} \\ &\geq \frac{1}{2}\min\left\{1, \lambda_c c^k (1-c)^{2pn-k}\right\}, \end{aligned}$$

where the last inequality follows from Lemma 21. □

With Lemmas 87 and 88 we are capable of proving the upper bounds on the expected runtime until the $(1+(\lambda,\lambda))$ GA escapes the local optimum. We estimate the runtime both in terms of the number of fitness evaluations and the number of iterations, denoted by T_F and T_I respectively.

Theorem 40. *Let $k \leq \frac{n}{4}$. Assume that $p \geq \frac{2k}{n}$ and q_ℓ is as defined in Lemma 24. Then the expected runtime of $(1+(\lambda,\lambda))$ GA on $\textsc{Jump}_k$ is*

$$E[T_I] \leq \frac{4}{q_\ell \min\left\{1, \lambda_m \left(\frac{p}{2}\right)^k\right\} \min\left\{1, \lambda_c c^k (1-c)^{2pn-k}\right\}}$$

iterations and

$$E[T_F] \leq \frac{4(\lambda_m+\lambda_c)}{q_\ell \min\left\{1, \lambda_m \left(\frac{p}{2}\right)^k\right\} \min\left\{1, \lambda_c c^k (1-c)^{2pn-k}\right\}}$$

fitness evaluations if the algorithm starts in the local optimum.

Proof. When the algorithm is in the local optimum it stays there until it moves to the optimum. During this time in each iteration it has the same probability P to move into the global optimum, which is the probability that a successful mutation phase is followed by a successful crossover phase:

$$P = p_m p_c \geq \frac{q_\ell}{2}\lambda_m \left(\frac{p}{2}\right)^k \cdot \frac{1}{2}\lambda_c c^k (1-c)^{2pn-k}.$$

Hence we obtain an expected optimization time in terms of iterations of

$$E[T_I] = \frac{1}{P} \leq \frac{4}{q_\ell \lambda_m \lambda_c \left(\frac{pc}{2}\right)^k (1-c)^{2pn-k}}.$$

In each iteration the $(1+(\lambda,\lambda))$ GA performs exactly $\lambda_m+\lambda_c$ fitness evaluations, which gives us an expected number of

$$E[T_F] \leq \frac{4(\lambda_m+\lambda_c)}{q_\ell \lambda_m \lambda_c \left(\frac{pc}{2}\right)^k (1-c)^{2pn-k}}$$

fitness evaluations in total. □

With help of Theorem 40 we deliver good values for the parameters, namely $p = c = \sqrt{\frac{k}{n}}$ and $\lambda_c = \lambda_m = \sqrt{\frac{n}{k}}^k$. We omit the proof that these parameters yield the lowest upper bound (apart from optimizing the $e^{O(k)}$ factor), since it is just a routine work with complicated derivatives, but we state the runtime bounds resulting from these settings in the following corollary. In order to use this result when we compute the runtime with the random initialization we also formulate this theorem for general population sizes.

Corollary 4. *Let $k \in [2..\lfloor \frac{n}{4} \rfloor]$. Assume that $p = c = \sqrt{\frac{k}{n}}$ and $\lambda_m = \lambda_c = \lambda \leq 2^k \sqrt{\frac{n}{k}}^k$. Then the expected runtime of $(1+(\lambda,\lambda))$ GA on $\textsc{Jump}_k$ is $E[T_F] \leq n^k k^{-k} e^{O(k)} \lambda^{-1}$ fitness evaluations and $E[T_I] \leq n^k k^{-k} e^{O(k)} \lambda^{-2}$ iterations. For $\lambda = \sqrt{\frac{n}{k}}^k$ these bounds are $E[T_I] \leq e^{O(k)}$ and $E[T_F] \leq \sqrt{\frac{n}{k}}^k e^{O(k)}$.*

Proof. With $\lambda \leq 2^k \sqrt{\frac{n}{k}}^k$ we have $\lambda(\frac{p}{2})^k \leq 1$ and $\lambda c^k (1-c)^{2pn-k} \leq 1$. Consequently, by Theorem 40 we have

$$E[T_I] \leq \frac{4}{q_\ell \lambda^2 \left(\frac{k}{2n}\right)^k \left(1-\sqrt{\frac{k}{n}}\right)^{2\sqrt{kn}-k}}$$

$$= \frac{2^{k+2}\left(\frac{n}{k}\right)^k}{q_\ell \lambda^2 \left(1-\sqrt{\frac{k}{n}}\right)^{2\sqrt{kn}-k}}$$
$$\leq \frac{2^{k+2}}{q_\ell \lambda^2}\left(\frac{n}{k}\right)^k \left(1-\sqrt{\frac{k}{n}}\right)^{-2\sqrt{kn}}$$
$$\leq \frac{2^{k+2}}{q_\ell \lambda^2}\left(\frac{n}{k}\right)^k \left(1-\sqrt{\frac{k}{n}}\right)^{-\sqrt{\frac{n}{k}}2k},$$

Where $q_\ell \in [0.1, 1]$ is a constant defined in Lemma 24. By the estimate $(1-x)^{-\frac{1}{x}} \leq 4$ which holds for all $x \in (0, \frac{1}{2}]$ and by $\sqrt{\frac{k}{n}} \leq \sqrt{\frac{n}{4n}} = \frac{1}{2}$ we have

$$\begin{aligned} E[T_I] &\leq \frac{2^{k+2}}{q_\ell \lambda^2}\left(\frac{n}{k}\right)^k \left(1-\sqrt{\frac{k}{n}}\right)^{-\sqrt{\frac{n}{k}}2k} \\ &\leq \frac{2^{k+2}}{q_\ell \lambda^2}\left(\frac{n}{k}\right)^k 4^{2k} \\ &= \left(\frac{n}{k}\right)^k \frac{e^{O(k)}}{\lambda^2}. \end{aligned} \tag{54}$$

The expected number of fitness evaluations is $\lambda_m + \lambda_c = 2\lambda$ times greater, hence we have

$$E[T_F] \leq \left(\frac{n}{k}\right)^k \frac{e^{O(k)}}{\lambda}. \tag{55}$$

Putting $\lambda = \sqrt{\frac{n}{k}}^k$ into (54) and (55) we have $E[T_I] \leq e^{O(k)}$ and $E[T_F] \leq \sqrt{\frac{n}{k}}^k e^{O(k)}$. □

In the following corollary we show a wide range of the parameters, which yield a better upper bound than the mutation-based algorithm (apart from the $e^{O(k)}$ factor) for the sub-linear jump sizes. We do not show it for $k = \Theta(n)$, since in this case the upper bound given by Corollary 4 is $e^{O(k)}$, which is not better than the runtime of best mutation-based EAs.

Corollary 5. *Let $k \geq 2$ and $k = o(n)$. Assume that $p = \omega(\frac{k}{n})$, $c = \omega(\frac{k}{n})$ and $pc = O(\frac{k}{n})$. Define $\alpha := \lambda_m(\frac{p}{2})^k$ and $\beta := \lambda_c c^k (1-c)^{2pn-k}$. If α and β are at most one and $\alpha = \omega((\frac{k}{nc})^k)$ and $\beta = \omega((\frac{2k}{pn})^k)$, then the expected number of fitness evaluations until the $(1+(\lambda,\lambda))$ GA reaches the global optimum starting from the local optimum of JUMP$_k$ is*

$$E[T_F] = o\left(\left(\frac{n}{k}\right)^k\right) e^{O(k)}.$$

Before we prove the corollary we shortly discuss how one can choose the parameters that give us $o\left(\left(\frac{n}{k}\right)^k\right) e^{O(k)}$ runtime with Corollary 5. First we should choose p. It can be any value which is $\omega(\frac{k}{n})$ and which is $o(1)$. Then with the chosen value of p we can choose any c which is on the one hand $\omega(\frac{k}{n})$, but on the other hand $O(\frac{k}{n}p^{-1})$. Note that the closer p is to $\Theta(1)$, the smaller the range for c (thus we could not choose $p = \Theta(1)$, since in this case we cannot simultaneously satisfy $c = \omega(\frac{k}{n})$ and $pc = O(\frac{k}{n})$).

After we determine p and c, we can choose λ_m and λ_c. For λ_m the upper bound for the possible range is $(\frac{p}{2})^{-k}$, which follows from condition $\alpha = \lambda_m(\frac{p}{2})^k \leq 1$. The lower bound for λ_m is $\omega((\frac{2k}{pcn})^k)$, which follows from the condition $\alpha = \omega((\frac{k}{nc})^k)$. For λ_c we have similarly obtained bounds, which are $c^{-k}(1-c)^{-(2pn-k)}$ and $\omega((\frac{2k}{pcn})^k)(1-c)^{-(2pn-k)}$.

Generally, the choice of the λ_m and λ_c should be made in such way that they were as close as possible to the inverse probabilities of creating a good offsprings in the mutation and crossover phases respectively. By Lemma 21 this choice yields a $\Theta(1)$ probability of a successful iteration. Any smaller population size reduces this probability (usually greater than it reduces the cost of one iteration), while any greater population size only increases the cost of each iteration without significantly increasing the success probability.

Proof of Corollary 5. Since α and β are at most one, the runtime given by Theorem 40 is simplified to

$$\begin{aligned} E[T_F] &\leq \frac{4(\lambda_m+\lambda_c)}{q_\ell \lambda_m \lambda_c \left(\frac{pc}{2}\right)^k (1-c)^{2pn-k}} \\ &= \frac{4}{q_\ell \alpha c^k (1-c)^{2pn-k}} + \frac{4}{q_\ell \beta \left(\frac{p}{2}\right)^k}. \end{aligned}$$

We want both terms to be $o((\frac{n}{k})^k)e^{O(k)}$. For the first term it is sufficient if the three following conditions hold. (i) $c^k = \omega((\frac{k}{n})^k)$, which holds if $c = \omega(\frac{k}{n})$. (ii) $(1-c)^{2pn-k} = e^{-O(k)}$. For this it is sufficient to have $pc = O(\frac{k}{n})$, since then we have

$$(1-c)^{2pn-k} = (1-c)^{\frac{1}{c}(2pcn-kc)} \geq \left(\frac{1}{4}\right)^{2pcn} = \left(\frac{1}{4}\right)^{O(k)} = e^{-O(k)}.$$

(iii) $\alpha = \omega((\frac{k}{nc})^k)$, since this implies

$$\frac{4}{q_\ell \alpha c^k (1-c)^{2pn-k}} = \frac{o\left(\left(\frac{nc}{k}\right)^k\right)}{c^k e^{-O(k)}} = o\left(\left(\frac{n}{k}\right)^k\right) e^{O(k)}.$$

For the second term it is enough that the following two conditions hold. (i) $(\frac{p}{2})^k = \omega((\frac{k}{n})^k)$, for which it is sufficient to have $p = \omega(\frac{k}{n})$. (ii) β should not be too small, namely, $\beta = \omega((\frac{2k}{pn})^k)$, since it implies

$$\frac{4}{q_\ell \beta \left(\frac{p}{2}\right)^k} = o\left(\left(\frac{pn}{2k}\right)^k\right)\left(\frac{p}{2}\right)^{-k} = o\left(\left(\frac{n}{k}\right)^k\right). \qquad \square$$

Without having the lower bounds we cannot claim that other parameter settings are worse than the proposed one. However, since we believe our bound to be asymptotically tight (apart from the $e^{O(k)}$ factor), we show that the standard parameter setting does not give us such a good upper bound.

Corollary 6. *Let $k \in [2..\lfloor \frac{n}{4} \rfloor]$. Assume that $p = \frac{\lambda}{n}$, $c = \frac{1}{\lambda}$ and $\lambda_m = \lambda_c = \lambda$ for some $\lambda \in [2k..n]$. Then the expected runtime of $(1+(\lambda,\lambda))$ GA on $\textsc{Jump}_k$ is $E[T_I] = O(2^k n^k \lambda^{-2})$ iterations and $E[T_F] = O(2^k n^k \lambda^{-1})$ fitness evaluations.*

Proof. Since the standard parameter setting with $\lambda \geq 2k$ satisfies the conditions of Theorem 40, we obtain

$$\begin{aligned} E[T_I] &\leq \frac{4}{q_\ell \lambda^2 \left(\frac{\lambda}{2n\lambda}\right)^k \left(1-\frac{1}{\lambda}\right)^{2\lambda-k}} \\ &\leq \frac{4(2n)^k}{q_\ell \lambda^2 \left(1-\frac{1}{\lambda}\right)^{2\lambda}} = O\left(\frac{(2n)^k}{\lambda^2}\right). \end{aligned}$$

In each iteration the $(1+(\lambda,\lambda))$ GA performs 2λ fitness evaluations, thus we have

$$E[T_F] = 2\lambda E[T_I] = O\left(\frac{(2n)^k}{\lambda}\right).$$

□

Reaching the Local Optimum

We showed that the $(1+(\lambda,\lambda))$ GA with non-standard parameters setting can find the global optimum of JUMP$_k$ much faster than any standard mutation-based algorithms if the algorithms are started in the local optimum. However, the non-standard parameter setting includes an unnaturally large population size, which makes each iteration costly. At the same time, there is no guarantee that we increase the fitness by much in one iteration, which makes us pay with many fitness evaluations before we reach the local optimum. Hence we question how much the runtime with this parameter setting increases when we start at a random bit string. In this section we show that slightly changing the parameters we can obtain the runtime which is only by a $\sqrt{n}$ factor greater than the runtime when we start in the local optimum. The main result of this section is the following theorem.

Theorem 41. *Let $k \leq \frac{n}{16}$. If $\lambda_m = \lambda_c = \frac{1}{\sqrt{n}}\sqrt{\frac{n}{k}}^k$ and $p = c = \sqrt{\frac{k}{n}}$, then the expected runtime of the $(1+(\lambda,\lambda))$ GA with any initialization on the JUMP$_k$ function is at most $\sqrt{n}\sqrt{\frac{n}{k}}^k e^{O(k)}$ fitness evaluations.*

To prove Theorem 41 we first analyse the runtime until the $(1+(\lambda,\lambda))$ GA reaches the local optimum of JUMP$_k$.

Theorem 42. *Let $\lambda_m = \lambda_c = \lambda \geq \frac{n}{k}$ and $p = c = \sqrt{\frac{k}{n}}$. Then the expected time until the $(1+(\lambda,\lambda))$ GA reaches the local optimum of JUMP$_k$ with $k \leq \frac{n}{16}$ is at most $E[T_I] = ne^{O(k)}$ iterations.*

The main challenge in the proof of this theorem is that an offspring close to the optimum in the mutation phase can lie in the fitness valley and thus it is not selected as x'. In this section we call the mutation phase *successful* if the winner has at least one zero-bit of x flipped to one. If such a bit exists, we call it *critical* and we call a mutation phase offspring which has such a bit and does not lie in the fitness valley *good*. We write p_m to denote the probability of a successful mutation phase. We call the crossover phase *successful* if the winner of the crossover phase has inherited the critical bit from x' and all other bits are not changed compared to x (hence, the

name "critical", since we need such bit to be taken from x' for a successful iteration). We write p_c to denote the probability of this event.

To prove Theorem 42 we show two auxiliary lemmas for the mutation and crossover phases respectively.

Lemma 89. *Let $k \leq \frac{n}{16}$. If $\lambda_m \geq \frac{\sqrt{n}}{k\sqrt{k}}$ and $\ell \in \left[\frac{\sqrt{nk}}{2}, \frac{3\sqrt{nk}}{2}\right]$, then $p_m = \Theta(1)$.*

Proof. We denote the current distance to the global optimum by $d(x)$. We distinguish two cases depending on this distance $d(x)$.

1) $d(x) \geq \frac{3\sqrt{nk}}{2} + k$ (long distance from the optimum),

2) $d(x) < \frac{3\sqrt{nk}}{2} + k$ (short distance from the optimum).

Long distance from the optimum. When $d(x) \geq \frac{3\sqrt{nk}}{2} + k$, we have a probability of zero to sample an individual in the fitness valley, since for this we require $\ell > \frac{3\sqrt{nk}}{2}$. Thus to obtain a fitter individual it is sufficient to generate an offspring with at least one of its zero-bits flipped to one. We denote the number of zero-bits flipped to one-bits by ℓ_0. Since we assumed $\ell \in [\frac{\sqrt{nk}}{2}, \frac{3\sqrt{nk}}{2}]$, we estimate the probability q_m to generate a good offspring as follows.

$$\begin{aligned} q_m &= \Pr\left[\ell_0 > 0 \,\middle|\, \ell \in \left[\frac{\sqrt{nk}}{2}, \frac{3\sqrt{nk}}{2}\right]\right] \\ &= \left(1 - \frac{\binom{\mathrm{OM}(x)}{\ell}}{\binom{n}{\ell}}\right) \geq 1 - \left(\frac{\mathrm{OM}(x)}{n}\right)^{\ell} \\ &\geq 1 - \left(\frac{n - \frac{3}{2}\sqrt{nk}}{n}\right)^{\frac{\sqrt{nk}}{2}} \geq \frac{1}{2}\min\left\{1, \frac{3k}{4}\right\} = \Omega(1), \end{aligned}$$

where the last inequality follows from Lemma 21. Therefore, $p_m = 1 - (1 - q_m)^{\lambda} \geq q_m = \Omega(1)$.

Short distance from the optimum. When $d(x) < \frac{3\sqrt{nk}}{2} + k$ there is a positive probability to have a mutant in the fitness valley. However, if the number ℓ_0 of zero-bits flipped is in $[1..\ell/2]$, then the offspring is guaranteed to be good. With the union bound we estimate the probability of this event as

$$q_m \geq 1 - \Pr[\ell_0 = 0] - \Pr\left[\ell_0 > \frac{\ell}{2}\right].$$

Having $k \leq \frac{n}{16}$ and $\mathrm{OM}(x) > n - \frac{3\sqrt{nk}}{2} - k \geq \frac{9n}{16}$, we can use Lemma 26, which yields

$$\Pr\left[\ell_0 \geq \frac{\ell}{2}\right] \leq e^{-\frac{\ell}{336}}.$$

Hence, since we have $\ell \geq \frac{\sqrt{nk}}{2}$, this probability is at most $\exp(-\frac{\sqrt{nk}}{672})$.

We estimate the probability that we have not flipped a single zero-bit in the same way as for the long distance from the optimum, but we have another upper bound on the current fitness.

$$\Pr[\ell_0 = 0] = \frac{\binom{\mathrm{OM}(x)}{\ell}}{\binom{n}{\ell}} \leq \left(\frac{n-k}{n}\right)^{\ell} \leq \left(1 - \frac{k}{n}\right)^{\frac{\sqrt{nk}}{2}}$$

$$= 1 - \left(1 - \left(1 - \frac{k}{n}\right)^{\frac{\sqrt{nk}}{2}}\right) \leq 1 - \frac{1}{2}\min\left\{1, \frac{k\sqrt{k}}{2\sqrt{n}}\right\}$$

where the last inequality follows from Lemma 21. Therefore, we have

$$\begin{aligned} q_m &\geq 1 - \Pr[\ell_0 = 0] - \Pr\left[\ell_0 > \frac{\ell}{2}\right] \\ &\geq \frac{1}{2}\min\left\{1, \frac{k\sqrt{k}}{2\sqrt{n}}\right\} - e^{-\frac{\sqrt{nk}}{672}} \geq \frac{1}{4}\min\left\{1, \frac{k\sqrt{k}}{2\sqrt{n}}\right\}, \end{aligned}$$

where the last inequality holds when n is at least some sufficiently large constant. If $\frac{k\sqrt{k}}{2n} > 1$, this probability is already $\Omega(1)$ and hence $p_m = \Omega(1)$.

Otherwise, by Lemma 21 we compute

$$p_m = 1 - (1 - q_m)^{\lambda} \geq \frac{1}{2}\min\{1, \lambda q_m\}.$$

Since $\lambda \geq \frac{\sqrt{n}}{k\sqrt{k}}$, we have $\lambda q_m \geq \frac{1}{2}$ and therefore, $p_m = \Omega(1)$.

Finally, we note that for constant n the probability q_m is still positive, and hence $\Omega(1)$. □

We proceed with a lemma for the crossover phase.

Lemma 90. *Let $k \leq \frac{n}{16}$. Assume that $c = \sqrt{\frac{k}{n}}$, $\lambda_c \geq \sqrt{\frac{n}{k}}$ and $\ell \in \left[\frac{\sqrt{nk}}{2}, \frac{3\sqrt{nk}}{2}\right]$, and there is at least one critical bit in x'. Then $p_c = e^{-O(k)}$.*

Proof. To have a successful crossover offspring it is sufficient to take one critical bit from x' and all other different bits from x. Thus the probability q_c of generating one superior crossover offspring is

$$\begin{aligned} q_c = c(1-c)^{\ell-1} &\geq \sqrt{\frac{k}{n}}\left(1 - \sqrt{\frac{k}{n}}\right)^{\frac{3\sqrt{nk}}{2}-1} \\ &\geq \sqrt{\frac{k}{n}}\left(1 - \sqrt{\frac{k}{n}}\right)^{\sqrt{\frac{n}{k}}\frac{3k}{2}} = \sqrt{\frac{k}{n}}e^{-\Theta(k)}. \end{aligned}$$

Since we need only one of the $\lambda_c \geq \sqrt{\frac{n}{k}}$ offspring to be superior, by Lemma 21 we have

$$p_c = 1 - (1 - q_c)^{\lambda_c} \geq \frac{1}{2}\min\left\{1, \lambda_c\sqrt{\frac{k}{n}}e^{-O(k)}\right\} = e^{-O(k)}.$$ □

Now we are in position to prove Theorem 42

Proof of Theorem 42. We denote the probability to increase fitness in one iteration by P and we estimate this probability as follows.

$$P \geq \Pr\left[\ell \in \left[\frac{pn}{2}, \frac{3pn}{2}\right]\right] \cdot p_m \cdot p_c.$$

By Lemmas 25, 89, and 90 we have

$$P \geq (1 - o(1)) \cdot \Omega(1) \cdot e^{-O(k)} = e^{-O(k)}.$$

Therefore the expected runtime (in terms of iterations) until the $(1 + (\lambda, \lambda))$ GA reaches the local optimum of JUMP_k is

$$E[T_I] \leq \sum_{i=0}^{n-k} \frac{1}{P} \leq ne^{O(k)}. \qquad \square$$

Finally, we prove the main result of this section, Theorem 41.

Proof of Theorem 41. By Theorems 4 and 42 the upper bound on the total number of fitness evaluations of the $(1 + (\lambda, \lambda))$ GA with random initialization is

$$E[T_F] \leq \lambda ne^{O(k)} + \sqrt{\frac{n}{k}}^k \frac{e^{O(k)}}{\lambda}.$$

With $\lambda = \frac{1}{\sqrt{n}}\sqrt{\frac{n}{k}}^k$, we have

$$\begin{aligned} E[T_F] &\leq \frac{1}{\sqrt{n}}\sqrt{\frac{n}{k}}^k ne^{\Theta(k)} + \sqrt{\frac{n}{k}}^k \frac{e^{\Theta(k)}}{\frac{1}{\sqrt{n}}\sqrt{\frac{n}{k}}^k} \\ &\leq \sqrt{n}\sqrt{\frac{n}{k}}^k e^{\Theta(k)} + \sqrt{n}\sqrt{\frac{n}{k}}^k e^{\Theta(k)} = \sqrt{n}\sqrt{\frac{n}{k}}^k e^{\Theta(k)}. \qquad \square \end{aligned}$$

We note that $\lambda = \frac{1}{\sqrt{n}}\sqrt{\frac{n}{k}}^k$ is the value which minimizes our upper bound apart from the $e^{\Theta(k)}$ factor. We omit the proof of this fact, since it trivially follows from the minimization of a function $f(x) = ax + \frac{b}{x}$ via analysis of its derivative.

4.4.2 Lower Bound

In this subsection we show that a deviation from the instance-specific optimal parameters setting significantly increases the runtime. The consequence is that when the parameter k is unknown, we are not likely to choose a good static parameter setting.

To analyze the negative effect of a wrong parameter choice we use the precise expression of the probability P to go from the local to the global optimum in one iteration, which is

$$P = \sum_{\ell=0}^{n} p_\ell p_m(\ell) p_c(\ell), \tag{56}$$

where p_ℓ is the probability to choose ℓ bits to flip, $p_m(\ell)$ is the probability of a successful mutation phase conditional on the chosen ℓ, and $p_c(\ell)$ is the probability of a successful crossover phase conditional on the chosen ℓ and on the mutation being successful.

Since $\ell \sim \mathrm{Bin}(n,p)$, we have $p_\ell = \binom{n}{\ell} p^\ell (1-p)^{n-\ell}$. The probability of a successful mutation depends on the chosen ℓ. If $\ell < k$, then it is impossible to flip all k zero-bits, hence $p_m(\ell) = 0$. For larger ℓ the probability to create a good offspring in a single application of the mutation operator is $q_m(\ell) = \binom{n-k}{\ell-k}/\binom{n}{\ell}$. If $\ell \in [k+1..2k-1]$ then any good offspring occurs in the fitness valley and has a worse fitness than any other offspring that is not good. Hence, in order to have a successful mutation we need all λ_m offspring to be good. Therefore, the probability of a successful mutation is $(q_m(\ell))^{\lambda_m}$. For $\ell = k$ and $\ell \geq 2k$ we are guaranteed to choose a good offspring as the winner of the mutation phase if there is at least one. Therefore, the mutation phase is successful with probability $p_m(\ell) = 1-(1-q_m(\ell))^{\lambda_m}$.

In the crossover phase we can create a good offspring only if $\ell \geq k$. For this we need to take all k bits which are zero in x from x', and then take all $\ell-k$ one-bits which were flipped from x. The probability to do so in one offspring is $q_c(\ell) = c^k(1-c)^{\ell-k}$. Since we create λ_c offspring and at least one of them must be superior to x, the probability of the successful crossover phase is $p_c(\ell) = 1-(1-c^k(1-c)^{\ell-k})^{\lambda_c}$.

Putting these probabilities into (56) we obtain

$$\begin{aligned} P = {} & \binom{n}{k} p^k (1-p)^{n-k} \left(1-\left(1-\binom{n}{k}^{-1}\right)^{\lambda_m}\right)\left(1-(1-c^k)^{\lambda_c}\right) \\ & + \sum_{\ell=k+1}^{2k-1} \binom{n}{\ell} p^\ell (1-p)^{n-\ell} \left(\frac{\binom{n-k}{\ell-k}}{\binom{n}{\ell}}\right)^{\lambda_m} \left(1-(1-c^k(1-c)^{\ell-k})^{\lambda_c}\right) \\ & + \sum_{\ell=2k}^{n} \binom{n}{\ell} p^\ell (1-p)^{n-\ell} \left(1-\left(1-\frac{\binom{n-k}{\ell-k}}{\binom{n}{\ell}}\right)^{\lambda_m}\right)\left(1-(1-c^k(1-c)^{\ell-k})^{\lambda_c}\right). \end{aligned}$$

Via this expression for P we compute the expected runtime in terms of iterations as $E[T_I] = P^{-1}$ and the expected runtime in terms of fitness evaluations as $E[T_f] = (\lambda_m+\lambda_c)P^{-1}$. It is hard estimate precisely the probability P and thus the expected runtime. Therefore, to show the critical influence of the parameters on the runtime, we compute $E[T_f]$ precisely for $n = 2^{20}$ and $k \in \{2^2, 2^4, 2^6\}$ and for different parameter values. We fix $\lambda_m = \lambda_c = \sqrt{\frac{n}{k}}^k$ and take $p = 2^\delta\sqrt{\frac{k}{n}}$ and $c = 2^{-\delta}\sqrt{\frac{k}{n}}$ for all $\delta \in [-\log_2(\sqrt{\frac{n}{k}})..\log_2(\sqrt{\frac{n}{k}})]$; these limits for δ guarantee that both p and c do not exceed 1. Note that we preserve the invariant $pcn = k$, since otherwise the expected Hamming distance between x and any crossover offspring (the "search radius" of the $(1+(\lambda,\lambda))$ GA) is not k, which makes it even harder to find the global optimum. The results of this computation are shown in Figure 12.

As one can see, there is a relatively small interval where losses in runtime are of a small constant factor (for $\delta = -1$ the runtime is even slightly better), but generally the runtime is increased by a $\Theta(2^{|\delta|k})$ factor. Therefore, in order to solve JUMP_k effectively with the $(1+(\lambda,\lambda))$ GA using the static parameters, one has to guess the value of k with a small relative error. In practice when we optimize some JUMP-like problem we usually cannot tell in advance the size of jump which we must perform to escape local optima. Therefore, when we do not know jump size k in advance, we need some method of dynamic choice of parameters to avoid missing the optimal one.

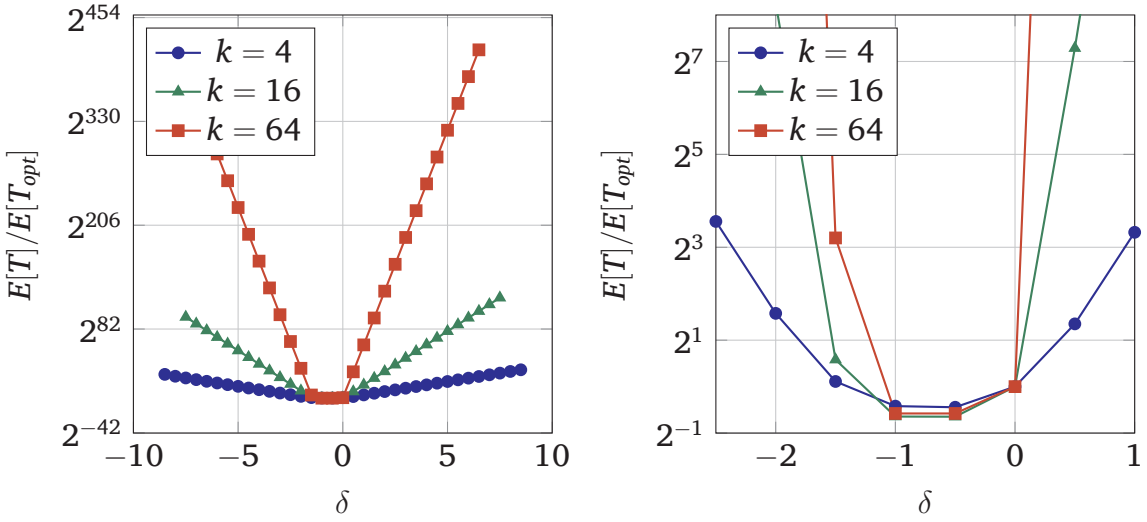


Figure 12 – *The ratio of the runtime with disturbed parameters to the runtime with the parameters suggested Corollary 4. The left plot shows the full picture for all considered values of δ. The right plot shows in more detail a smaller interval around the best values*

4.5 The Runtime Analysis of the Heavy-Tailed $(1 + (\lambda, \lambda))$ GA JUMP

Corollary 6 showed that the standard parameters of the $(1 + (\lambda, \lambda))$ GA are not very useful on jump, while taking mutation rate p and crossover bias c equal can be a good idea when making progress is difficult. Parameterizing $p = c = \sqrt{s/n}$, we obtain that an offspring after mutation and crossover has an expected Hamming distance of s from the parent. Hence the parameter s, in a similar manner as the mutation rate in a traditional mutation-based algorithm, quantifies the typical search radius of the $(1 + (\lambda, \lambda))$ GA. Hence, it is natural to choose s from the power-law distribution, since intuitively we should have the same effect as in [57] of having a good probability to choose s close to its optimal value k. However, if we try to use Corollary 4 and to set the population size λ according to our choice of s, namely $\lambda = (\frac{n}{s})^{s/2}$, then overestimating s will be very costly. For this reason we find it more natural to choose it randomly form another power-law distribution.

We conduct a mathematical runtime analysis of the $(1 + (\lambda, \lambda))$ GA with heavy-tailed choices of s and λ from a broad range of power-law distributions. It shows that for a power-law exponent $\beta_s > 1$ for the choice of s and a power-law exponent β_λ equal to two or slightly above, a very good performance can be obtained. The resulting runtimes are slightly higher than those stemming from the best, instance-specific static parameters, but still much below the runtimes of classic evolutionary algorithms.

While undoubtedly we have obtained parameters that work uniformly well over all jump functions, we also feel that our choices of the power-law exponent are quite natural, so that the name parameterless $(1 + (\lambda, \lambda))$ GA might be justified. There is not much to say on the choice of s, where apparently all power-laws (with exponent greater than one, which is a very natural assumption for any use of a power-law) give good results. For the choice of λ, we note that the cost of one iteration of the $(1 + (\lambda, \lambda))$ GA is 2λ fitness evaluations. Hence $2E[\lambda]$ is the expected cost of an iteration with a random choice of λ. Now any power-law exponent

$\beta_\lambda > 2$ gives a constant value for $E[\lambda]$. The larger β_λ is, the more the power-law distribution is concentrated on constant values. For constant λ, however, the $(1+(\lambda,\lambda))$ GA cannot profit a lot from the intermediate selection step, and thus shows a behavior similar to classic mutation-based algorithms. For this reason, choosing a power-law exponent rather close to two appears to be a natural choice. Based both on this informal argument and our mathematical results, for a practical application of our algorithm we recommend to use β_s slightly above one, say 1.1, and β_λ slightly above two, say 2.1.

The asymptotically best choice of β_λ (in the sense that the worst-case price for being instance-independent is lowest) is obtained from taking $\beta_\lambda = 2$. Since this alone would give an infinite value for $E[\lambda]$, one needs to restrict the range of values this distribution is defined on. To obtain an $O(nk^{\beta_s-1})$ price of instance-independence, a generous upper bound of 2^n is sufficient. To obtain our best price of instance-independence of $O(n\log n)$, a similar trick is necessary for the choice of s, namely taking $\beta_s = 1$ and capping the range at the (trivial) upper bound $s \leq n$. While we think that these considerations are interesting from the theoretical perspective as they explore the limits of our approach, we do not expect these hyperparameter choices to be useful in many practical applications. We note the runtime of the $(1+1)$ EA with heavy-tailed mutation rate was shown [57] to exceed the instance-specific best runtime of the $(1+1)$ EA by a factor of $\Theta(n^{\beta-0.5})$. Hence a power-law exponent β as low as possible (but larger than one) looks best from the theoretical perspective. In contrast, in the experiments in [57], no improvement was seen from lowering β below 1.5.

4.5.1 The Heavy-Tailed $(1+(\lambda,\lambda))$ GA with Multiple Parameter Choices

We now define the heavy-tailed $(1+(\lambda,\lambda))$ GA as motivated in the introduction. The main difference from the standard $(1+(\lambda,\lambda))$ GA is that at the start of each iteration the mutation rate p, the crossover bias c, and the population sizes λ_m and λ_c for the mutation and crossover phases are randomly chosen as follows. We sample $s \sim \text{pow}(\beta_s, u_s)$ and take $p = c = (\frac{s}{n})^{1/2}$. The population sizes are chosen via $\lambda_m = \lambda_c = \lambda \sim \text{pow}(\beta_\lambda, u_\lambda)$. Here the upper limits u_λ and u_s can be any positive integers and the power-law exponents β_λ and β_s can be any non-negative real numbers. We call these parameters of the power-law distribution the *hyperparameters of the heavy-tailed* $(1+(\lambda,\lambda))$ *GA* and we give recommendations on how to choose these hyperparameters. The pseudocode of this algorithm is shown in Algorithm 6. We note that it is not necessary to store the whole offspring populations, since only the best individual has a chance to be selected as mutation or crossover winner. Hence also large values for λ are algorithmically feasible.

4.5.2 Runtime Analysis

In this section we conduct a rigorous runtime analysis for our heavy-tailed $(1+(\lambda,\lambda))$ GA optimizing jump functions with jump size $k \in [2..\frac{n}{4}]$. We cover the full spectrum of the algorithm's hyperparameters $\beta_s, u_s, \beta_\lambda, u_\lambda$. For large ranges of the hyperparameters, in particular,

Algorithm 6 – The heavy-tailed $(1+(\lambda,\lambda))$ GA with multiple parameter choices maximizing a pseudo-Boolean function f

$x \leftarrow$ random bit string of length n
while not terminate **do**
  Choose $s \sim \text{pow}(\beta_s, u_s)$
  $p \leftarrow (\frac{s}{n})^{1/2}$
  $c \leftarrow (\frac{s}{n})^{1/2}$
  Choose $\lambda \sim \text{pow}(\beta_\lambda, u_\lambda)$
Mutation phase:
  Choose $\ell \sim \text{Bin}(n,p)$
  for $i \in [1..\lambda]$ **do**
    $x^{(i)} \leftarrow$ a copy of x
    Flip ℓ bits in $x^{(i)}$ chosen uniformly at random
  end for
  $x' \leftarrow \arg\max_{z \in \{x^{(1)},\dots,x^{(\lambda)}\}} f(z)$
Crossover phase:
  for $i \in [1..\lambda]$ **do**
    Create $y^{(i)}$ by taking each bit from x' with probability c and from x with probability $(1-c)$
  end for
  $y \leftarrow \arg\max_{z \in \{y^{(1)},\dots,y^{(\lambda)}\}} f(z)$
  if $f(y) \geq f(x)$ **then**
    $x \leftarrow y$
  end if
end while

for natural values like $\beta_s = \beta_\lambda = 2+\varepsilon$ and $u_s = u_\lambda = \infty$, we observe a performance that is only a little worse than the one with the best instance-specific static parameters. This price of instance-independence can be brought down to an $O(n\log(n))$ factor. Taking into account the effect of failing to guess the optimal parameters shown in Section 4.4.2, this is a fair price for a one-size-fits-all algorithm.

Since a typical optimization process on jump functions consists of two very different regimes, we analyze separately the difficult regime of going from the local optimum to the global one and the easy ONEMAX-style regime encountered before that.

Escaping the Local Optimum

The time to leave the local optimum (necessarily to the global one) is described in the following theorem and Table 4. We will see later that unless $\beta_\lambda < 2$, and this is not among our recommended choices, or $k = 2$, the time to reach the local optimum is not larger than the time to go from the local to the global optimum. Hence for $\beta_\lambda \geq 2$, the table also gives valid runtime estimates for the complete runtime.

β_λ	$E[T_f]p_s$ if $u_\lambda < \left(\frac{n}{k}\right)^{k/2}$	$E[T_f]p_s$ if $u_\lambda \geq \left(\frac{n}{k}\right)^{k/2}$
$[0,1)$	$e^{\Theta(k)}\frac{1}{u_\lambda}\left(\frac{n}{k}\right)^k$	$u_\lambda e^{\Theta(k)}$
$=1$		$u_\lambda e^{\Theta(k)}/\left(1+\ln\left(u_\lambda\left(\frac{n}{k}\right)^{k/2}\right)\right)$
$(1,2)$		$e^{\Theta(k)}u_\lambda^{2-\beta}\left(\frac{n}{k}\right)^{k/2(\beta-1)}$
$=2$	$e^{\Theta(k)}\frac{\ln(u_\lambda+1)}{u_\lambda}\left(\frac{n}{k}\right)^k$	$e^{\Theta(k)}\ln(u_\lambda)\left(\frac{n}{k}\right)^{k/2}$
$(2,3)$	$e^{\Theta(k)}\frac{1}{u_\lambda^{3-\beta}}\left(\frac{n}{k}\right)^k$	$e^{\Theta(k)}\left(\frac{n}{k}\right)^{k/2(\beta-1)}$
$=3$	$e^{\Theta(k)}\frac{1}{\ln(u_\lambda+1)}\left(\frac{n}{k}\right)^k$	$e^{\Theta(k)}\left(\frac{n}{k}\right)^k/\ln\left(\left(\frac{n}{k}\right)^k\right)$
>3	$e^{\Theta(k)}\left(\frac{n}{k}\right)^k$	

Table 4 – *Influence of the four hyperparameters $\beta_s, u_s, \beta_\lambda, u_\lambda$ on the expected number $E[T_f]$ of fitness evaluations the heavy-tailed $(1+(\lambda,\lambda))$ GA starting in the local optimum takes to optimize JUMP$_k$. Since all runtime bounds are of type $E[T_f] = F(\beta_\lambda, u_\lambda)/p_s$, where $p_s = \Pr[s \in [k..2k]]$, to ease reading we only state $F(\beta_\lambda, u_\lambda) = E[T_f]p_s$. By taking $\beta_s = 1+\varepsilon$ or $\beta_s = 1 \wedge u_s = n$, one obtains $p_s = k^\varepsilon$ or $p_s = O(\log n)$. Using $\beta_\lambda = 2$ and an exponential u_λ gives the lowest price of an $O(n \log n)$ factor for being independent of the instance parameter k. We also advertise the slightly inferior combination $\beta_\lambda = 2+\varepsilon$ and $u_\lambda = +\infty$ as for $\beta_\lambda > 2$ each iteration has a constant expected cost and u_λ has no influence on the runtime (if chosen large enough). If $\beta_\lambda \geq 2$ and $k \geq 3$, then the times stated are also the complete runtimes starting from a random initial solution*

Theorem 43. *Let $k \in [2..\frac{n}{4}]$. Assume that we use the heavy-tailed $(1+(\lambda,\lambda))$ GA (Algorithm 6) to optimize JUMP$_k$, starting already in the local optimum. Then the expected number of the fitness evaluations until the optimum is found is shown in Table 4, where p_s denotes the probability that $s \in [k..2k]$. If $u_s \geq 2k$, then p_s is*

- $\Theta((\frac{k}{u_s})^{1-\beta_s})$, *if* $\beta_s \in [0,1)$,
- $\Theta(\frac{1}{\ln(u_s)})$, *if* $\beta_s = 1$, *and*
- $\Theta(k^{\beta_s-1})$, *if* $\beta_s > 1$.

Before the proof we distill the following recommendations on how to set the parameters of the power-law distributions from Theorem 43.

Distribution of λ: We note that when guessing u_λ right (depending on k), and only then, good runtimes can be obtained for $\beta_\lambda < 2$. Since we aim at a (mostly) parameterless approach, this is not very interesting. When $\beta_\lambda > 3$, we observe a slow runtime behavior similar to the one of the $(1+1)$ EA with heavy-tailed mutation rate [57]. This is not surprising since with this distribution of λ typically only small values of λ are sampled. We profit most from the strength of the heavy-tailed $(1+(\lambda,\lambda))$ GA when β_λ is close to two. If it is larger than two, then each iteration has an expected constant cost, so we can conveniently choose $u_\lambda = \infty$ without that this can have a negative effect on the runtime. This is a hyperparameter setting we would recommend as a first, low-risk attempt to use this algorithm. Slightly better results are obtained from using $\beta_\lambda = 2$. Now a finite value for u_λ is necessary, but the logarithmic influence of u_λ on the runtime allows to be generous, e.g., taking u_λ exponential in n. Smaller values lead to minimally better runtimes as long as one stays above the boundary $(\frac{n}{k})^{k/2}$, so optimizing here is risky.

Distribution of s: The distribution of s is less critical as long as $u_s \geq 2k$. Aiming at an algorithm free from critical parameter choices, we therefore recommend to take $u_s = n$ unless

there is a clear indication that only short moves in the search space are necessary. Once we decided on $u_s = n$, a β_s value below one is not interesting (apart from very particular situations). Depending on what jump sizes we expect to encounter, taking $\beta_s = 1$ leading to an $O(\log n)$-factor contribution of s to the runtime or taking $\beta_s = 1 + \varepsilon$, $\varepsilon > 0$ but small, leading to an $O(k^\varepsilon)$-factor contribution to the runtime are both reasonable choices.

For the proof we also need the following supplementary lemma, which can easily be deduced from Lemmas 87 and 88.

Lemma 91. *Let $\lambda_m = \lambda_c = \lambda$ and $p = c = (\frac{s}{n})^{1/2}$ with $s \in [k..2k]$. If the current individual x of the $(1 + (\lambda, \lambda))$ GA is in the local optimum of $\textsc{Jump}_k$, then the probability that the algorithm finds the global optimum in one iteration is at least $e^{-\Theta(k)} \min\{1, (\frac{k}{n})^k \lambda^2\}$.*

Proof of Theorem 43. Let F be the event that the algorithm finds the global optimum in one iteration when the current individual x is already in the local optimum. The probability P of this event is at least

$$P \geq p_{(F|s)} p_s,$$

where $p_{(F|s)} = \Pr[F \mid s \in [k..2k]]$ and $p_s = \Pr[s \in [k..2k]]$. The expected number of iterations T_I until we find the optimum is therefore

$$E[T_I] = \frac{1}{P} \leq \frac{1}{p_{(F|s)} p_s}.$$

In each iteration the heavy-tailed $(1 + (\lambda, \lambda))$ GA performs 2λ fitness evaluation (where λ is chosen from a power-law distribution at the start of the iteration). Using Wald's equation (Lemma 20) we compute the expected runtime T_f in terms of fitness evaluations from T_I.

$$E[T_f] = E[T_I] E[2\lambda] = \frac{E[2\lambda]}{P} \leq \frac{2E[\lambda]}{p_{(F|s)} p_s}.$$

In the remainder of the proof we estimate how $E[\lambda]$, $p_{(F|s)}$, and p_s depend on the hyperparameters of the algorithm.

The expected value of λ is

$$E[\lambda] = \sum_{i=1}^{u_\lambda} p_\lambda(i) i = C_{\beta_\lambda, u_\lambda} \sum_{i=1}^{u_\lambda} i^{1-\beta_\lambda},$$

where $p_\lambda(i) = \Pr[\lambda = i]$. We compute the conditional probability of F as

$$p_{(F|s)} = \sum_{i=1}^{u_\lambda} p_\lambda(i) p_{(F|s,\lambda)}(i),$$

where $p_{(F|s,\lambda)}(i) = \Pr[F \mid s \in [k..2k] \wedge \lambda = i]$. Note that event $\lambda = i$ does not depend on the choice of s. By Lemma 91 we have

$$p_{(F|s,\lambda)}(i) \geq \begin{cases} \left(\frac{k}{n}\right)^k i^2 e^{-\Theta(k)}, & \text{if } i \leq \left(\frac{n}{k}\right)^{k/2}, \\ e^{-\Theta(k)}, & \text{otherwise.} \end{cases}$$

We consider two cases of the size of u_λ relative to k and n. First, if $u_\lambda < (\frac{n}{k})^{k/2}$, then we have

$$p_{(F|s)} \geq \sum_{i=1}^{u_\lambda} C_{\beta_\lambda,u_\lambda} i^{-\beta_\lambda} \left(\frac{k}{n}\right)^k i^2 e^{-\Theta(k)} = C_{\beta_\lambda,u_\lambda} e^{-\Theta(k)} \left(\frac{k}{n}\right)^k \sum_{i=1}^{u_\lambda} i^{2-\beta_\lambda}.$$

Hence, we have

$$E[T_f] = \frac{C_{\beta_\lambda,u_\lambda} \sum_{i=1}^{u_\lambda} i^{1-\beta_\lambda}}{p_s C_{\beta_\lambda,u_\lambda} e^{-\Theta(k)} \left(\frac{k}{n}\right)^k \sum_{i=1}^{u_\lambda} i^{2-\beta_\lambda}} = p_s^{-1} e^{\Theta(k)} \left(\frac{n}{k}\right)^k \frac{\sum_{i=1}^{u_\lambda} i^{1-\beta_\lambda}}{\sum_{i=1}^{u_\lambda} i^{2-\beta_\lambda}}.$$

In the second case, if $u \geq (\frac{n}{k})^{k/2}$, we have

$$\begin{aligned} p_{(F|s)} &\geq \sum_{i=1}^{\lfloor (\frac{n}{k})^{k/2} \rfloor} C_{\beta_\lambda,u_\lambda} i^{-\beta_\lambda} \left(\frac{k}{n}\right)^k i^2 e^{-\Theta(k)} + \sum_{\lfloor (\frac{n}{k})^{k/2} \rfloor + 1}^{u_\lambda} C_{\beta_\lambda,u_\lambda} i^{-\beta_\lambda} e^{-\Theta(k)} \\ &= C_{\beta_\lambda,u_\lambda} e^{-\Theta(k)} \left(\left(\frac{k}{n}\right)^k \sum_{i=1}^{\lfloor (\frac{n}{k})^{k/2} \rfloor} i^{2-\beta_\lambda} + \sum_{i=\lfloor (\frac{n}{k})^{k/2} \rfloor + 1}^{u_\lambda} i^{-\beta_\lambda} \right). \end{aligned}$$

Therefore,

$$E[T_f] \leq \frac{2E[\lambda]}{p_{(F|s)} p_s} \leq \frac{e^{\Theta(k)} \sum_{i=1}^{u_\lambda} i^{1-\beta_\lambda}}{p_s \left(\left(\frac{k}{n}\right)^k \sum_{i=1}^{\lfloor (\frac{n}{k})^{k/2} \rfloor} i^{2-\beta_\lambda} + \sum_{i=\lfloor (\frac{n}{k})^{k/2} \rfloor + 1}^{u_\lambda} i^{-\beta_\lambda} \right)}.$$

Viewing these two cases together, we obtain

$$E[T_f] \leq \frac{e^{\Theta(k)} S_1}{p_s \left(\left(\frac{k}{n}\right)^k S_2 + S_0 \right)}, \tag{57}$$

where

- $S_1 := \sum_{i=1}^{u_\lambda} i^{1-\beta_\lambda}$,
- $S_2 := \sum_{i=1}^{\min\{\lfloor (\frac{n}{k})^{k/2}, u_\lambda \rfloor} i^{2-\beta_\lambda}$, and
- $S_0 := \sum_{i=\lfloor (\frac{n}{k})^{k/2} \rfloor + 1}^{u_\lambda} i^{-\beta_\lambda}$ if $u_\lambda > (\frac{n}{k})^{k/2}$ and $S_0 := 0$ otherwise.

Table 5 shows the estimates of S_1, S_2 and S_0, which follow from Lemma 2. We also note that the estimates for $p_s = C_{\beta_s,u_s} \sum_{i=k}^{2k} i^{-\beta}$ follow from Lemmas 2 and 5. We omit these elementary calculations. Putting these estimates into (57) proves the theorem. □

Reaching the Local Optimum

In this section we show that the heavy-tailed choice of parameters lets the $(1+(\lambda,\lambda))$ GA reach the local optimum relatively fast. Without proof, we note that if $\beta_\lambda \geq 2$ and $k \geq 3$, then the time to reach the local optimum is not larger than the time to go from the local to the global optimum. For a set of hyperparameters giving the best price for instance-independence, we now show an $O(n^2 \log^2(n))$ time bound for reaching the local optimum.

β_λ	S_1	S_2	S_0 if $u_\lambda > (\frac{n}{k})^{k/2}$
$[0,1)$	$\Theta(u_\lambda^{2-\beta_\lambda})$	$\Theta\left((\min\{u_\lambda, (\frac{n}{k})^{k/2}\})^{3-\beta_\lambda}\right)$	$\Theta\left(u_\lambda^{1-\beta_\lambda} - (\frac{n}{k})^{k(1-\beta_\lambda)/2}\right)$
$=1$			$\Theta\left(\ln\left(u_\lambda(\frac{k}{n})^{k/2}\right)\right)$
$(1,2)$			$\Theta\left((\frac{n}{k})^{k(1-\beta_\lambda)/2} - u_\lambda^{1-\beta_\lambda}\right)$
$=2$	$\Theta(\log(u_\lambda))$		
$(2,3)$	$\Theta(1)$		
$=3$		$\Theta\left(\log(\min\{u_\lambda, (\frac{n}{k})^{k/2}\})\right)$	
>3		$\Theta(1)$	

Table 5 – *The values of S_1, S_2 and S_0 used in the proof of Theorem 43*

Theorem 44. *Let $u_\lambda = 2^{\Theta(n)}$, $\beta_\lambda = 2$, $u_s = \Theta(n)$, and $\beta_s = 1$. Then the expected runtime until the heavy-tailed $(1+(\lambda,\lambda))$ GA reaches the local optimum of $\textsc{Jump}_k$ starting in a random string is at most $O(n^2\log^2(n))$ fitness evaluations. For greater β_λ and any u_λ this runtime is at most $O(n\log^2(n))$. In both cases with $\beta_s > 1$ and any $u_s \in \mathbb{N}$ the runtime is reduced by a $\Theta(\log(n))$ factor.*

Proof. We prove the theorem only for $\beta_\lambda = 2$ and $\beta_s = 1$, since for other hyperparameter values the arguments are identical. By Lemma 5, the probability $p_{s,\lambda}$ to choose $s = 1$ and $\lambda = 1$ is

$$p_{s,\lambda} = C_{\beta_s,u_s}1^{(-1)}C_{\beta_\lambda,u_\lambda}1^{(-2)} = \Theta\left(\frac{1}{\log(n)}\right).$$

With $s = 1$ and $\lambda = 1$ the algorithm essentially performs an iteration of the $(1+1)$ EA with mutation rate $\frac{1}{n}$, since there is no selection of the mutation winner and each bit of the crossover offspring is flipped with probability $\sqrt{\frac{1}{n}}^2 = \frac{1}{n}$. Therefore, if the algorithm has not reached the local optimum, then the probability P to have a true progress in one iteration is at least

$$P \geq p_{s,\lambda}\frac{n-i}{n},$$

where i is the current fitness of x. Therefore, by Lemma 2 the expected number of iterations until the algorithm reaches the local optimum is at most

$$E[T_I] \leq \sum_{i=0}^{n-k-1}\frac{n}{p_{s,\lambda}(n-i)} \leq \Theta(\log(n))\cdot n\cdot O(\log(n)) = O(n\log^2(n)).$$

Since by Lemma 6 the expected number of fitness evaluations per iteration is $\Theta(\log(u_\lambda)) = \Theta(n)$, by Wald's equation (Lemma 20) we have

$$E[T_f] = E[T_I]E[2\lambda] \leq O(n\log^2(n))\cdot\Theta(n) = O(n^2\log^2(n)).$$

□

4.6 Conclusion of Chapter 4

In Section 4.1 we performed the first runtime analysis of a crossover-based algorithm using the heavy-tailed parameter choice and observed that the fast mutation operator not only

can relieve the algorithm designer from the task of choosing a suitable mutation rate, but it can also lead to runtimes asymptotically better than any static choice of the mutation rate. However, different from previous works, where any power-law exponent greater than one could be used, our work requires that β is between 2 and 3 and thus shows that, different from what the previous works suggest, the choice of β can be non-trivial.

On the technical side, our work shows that algorithms with a heavy-tailed number of offspring can be much easier to analyze than those with a self-adjusting number of offspring (such as the self-adjusting $(1 + (\lambda, \lambda))$ GA [31]), since Wald's equation allows to estimate the expected runtime by the product of the expected number of iterations and the expected number of offspring generated in one iteration. This also gives a clear hint on suitable values for the power-law exponent. (Only) by taking $\beta > 2$, we can ensure that the expected number of offspring generated per iteration is constant.

In Section 4.3, we have shown that the $(1 + (\lambda, \lambda))$ GA optimizes the LEADINGONES function in $\Theta(n^2/\lambda)$ parallel time and $\Theta(n^2)$ total runtime, regardless of the parameter λ. This shows that, not surprisingly, the $(1 + (\lambda, \lambda))$ GA here does not profit from the better exploration mechanism (as it did on ONEMAX and random satisfiability instances). At the same time, however, it still achieves asymptotically the same parallel runtime as the $(1 + \lambda)$ EA and the same total runtime as any $(1 + \lambda)$ EA with $\lambda \leq n$.

In Section 4.4 we performed the first runtime analysis of the $(1 + (\lambda, \lambda))$ GA on a multimodal problem and we observed that this algorithm also has a runtime advantage over classic algorithms on multimodal objective functions, and a much more pronounced one. Whereas the advantage in the previous results on unimodal problems was a gain of a logarithmic factor, we have shown here a runtime that is almost the square root of the runtime of classic algorithms.

For the $(1 + (\lambda, \lambda))$ GA to show such a good performance, its parameters have to be chosen differently from what was suggested in previous works, in particular, the mutation rate and crossover bias have to be larger. We developed some general suggestions in Corollary 5 that might ease the future use of this algorithm.

TO solve the problem of crucial dependency of optimal static parameters on the problem instance, in Section 4.5 we proposed a variant of the $(1 + (\lambda, \lambda))$ GA with a heavy-tailed choice of both the population size λ and the search radius s. To the best of our knowledge, this is the first time that two parameters of an EA are chosen in this manner. Our mathematical runtime analysis showed that this algorithm with suitable, but natural choices of the distribution parameters can optimize all jump functions in a time that is only mildly higher than the runtime of the $(1 + (\lambda, \lambda))$ GA with the best instance-specific parameter values.

We are optimistic that the insights gained on the jump functions benchmark extend, at least to some degree, also to other non-unimodal problems. Clearly, supporting this hope with rigorous results is an interesting direction for future research. From a broader perspective, this work suggests to try to use heavy-tailed parameter choices for more than one parameter simultaneously. Our rigorous results indicate that the prices for ignorant (heavy-tailed) choices of parameters simply multiply. For a small number of parameters with critical influence on the performance, this might be a good deal.

Conclusion

This thesis makes several contributions to the field of the theory of evolutionary computation. First, it proposes the following new analysis methods.

- The method of complete trees.
- The method of drift analysis of no-drift processes.
- The method of two Markov chains for the analysis on plateaus.
- The additive drift theorem with tail bounds.

With these methods we performed an analysis of various population-based EAs and delivered some totally new insights about the working principles of EAs. Although some of our methods are problem-specific, we are optimistic that they need only small modifications to be applied to a more general problems. We also believe that our methods can be combined to a more powerful tools. For example, the method of the analysis of EAs on plateaus can be combined with the ideas of the complete trees to deliver the runtime of the $(\mu + \lambda)$ EA on plateaus, but we leave it as an open topic for the further research.

Among the results obtained with the developed methods, we find the following ones most notable.

- The asymptotically tight precise bounds on the runtime of the $(\mu + \lambda)$ EA and the (μ, λ) EA on ONEMAX, which show in details the interplay of the parent and offspring populations sizes for these algorithms. In particular, we observed that the $(\mu + \lambda)$ EA makes progress after filling the top fitness level with some number of the best individuals and that the convergence of the (μ, λ) EA crucially depends on the absolute population sizes.
- The runtime results for the $(1 + 1)$ EA and $(\lambda \overset{1:1}{+} \lambda)$ EA on PLATEAU functions, which showed that the choice of the mutation operator does not have a significant influence on the behavior of these EAs on plateaus.
- The runtime of the $(1 + (\lambda, \lambda))$ GA with variable λ on LEADINGONES, where we use the argument of the average drift per fitness evaluation for an algorithm with a variable cost of one iteration.

We also proposed a new algorithm, the heavy-tailed $(1 + (\lambda, \lambda))$ GA, and showed that a simple strategy of choosing parameters randomly (but from a well-chosen distribution) can not only relieve us from choosing the right static parameters for a small price (as it does on JUMP), but also gives us the better runtime than the one which could be obtained with static parameters (as it does on ONEMAX).

To designate the further research directions, we find it profitable to extend the proposed methods for more complicated crossover-based genetic algorithms such as the $(\mu + \lambda)$ GA. At the moment we find it very challenging, since it would require more complex structures than in the complete trees which take into account the genotype of two parents of each individual.

We also find the approach of the random parameter selection worth the further investigation to find on which problems it can be beneficial. Clearly, there are two situations when it might either reduce the runtime (as for the $(1 + (\lambda, \lambda))$ GA on ONEMAX) or produce a one-size-fits-all algorithm (as for the $(1 + 1)$ EA [57] and the $(1 + (\lambda, \lambda))$ GA on JUMP). However, there are also functions for which missing the optimal parameters range can be destructive and results into a drift outwards the optimum, e.g., the HOTTOPIC functions [100]. Therefore, it is important to determine the scope in which this method of dynamic parameters choice helps.

Acknowledgemnts

This thesis would be impossible without the support of my parents and of my muse and wife Rita who always were with me during my long way to the PhD defense. I am also very thankful to all my teachers who showed me the beauty of the mathematical view of this world: Alexey Golovin, Nikolay Dodonov, Maxim Buzdalov and Benjamin Doerr.

References

[1] Fawaz Alanazi and Per Kristian Lehre. “Runtime analysis of selection hyper-heuristics with classical learning mechanisms”. In: *Congress on Evolutionary Computation, CEC 2104*. IEEE, 2014, pp. 2515–2523.

[2] Denis Antipov, Maxim Buzdalov, and Benjamin Doerr. “First Steps Towards a Runtime Analysis When Starting with a Good Solution”. In: *Parallel Problem Solving from Nature, PPSN 2020, Part II*. Springer, 2020, pp. 560–573.

[3] Denis Antipov, Arina Buzdalova, and Andrew Stankevich. “Runtime analysis of a population-based evolutionary algorithm with auxiliary objectives selected by reinforcement learning”. In: *Genetic and Evolutionary Computation Conference, GECCO 2018, Companion Material*. ACM, 2018, pp. 1886–1889.

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

[5] Thomas Bäck. “Optimal mutation rates in genetic search”. In: *International Conference on Genetic Algorithms, ICGA 1993*. Morgan Kaufmann, 1993, pp. 2–8.

[6] Golnaz Badkobeh, Per Kristian Lehre, and Dirk Sudholt. “Unbiased Black-Box Complexity of Parallel Search”. In: *Parallel Problem Solving from Nature, PPSN 2014*. Springer, 2014, pp. 892–901.

[7] Süntje Böttcher, Benjamin Doerr, and Frank Neumann. “Optimal Fixed and Adaptive Mutation Rates for the LeadingOnes Problem”. In: *Parallel Problem Solving from Nature, PPSN 2010*. Springer, 2010, pp. 1–10.

[8] Dimo Brockhoff et al. “On the Effects of Adding Objectives to Plateau Functions”. In: *IEEE Transactions on Evolutionary Computation* 13 (2009), pp. 591–603.

[9] M. Buzdalov, A. Buzdalova, and A. Shalyto. “A first step towards the runtime analysis of evolutionary algorithm adjusted with reinforcement learning”. In: *Proceedings of the International Conference on Machine Learning and Applications*. Vol. 1. 2013, pp. 203–208.

[10] Maxim Buzdalov and Arina Buzdalova. “OneMax helps optimizing XdivK: Theoretical runtime analysis for RLS and EA + RL”. In: *Proceedings of Genetic and Evolutionary Computation Conference Companion*. 2014, pp. 201–202.

[11] Maxim Buzdalov and Benjamin Doerr. “Runtime analysis of the $(1 + (\lambda, \lambda))$ genetic algorithm on random satisfiable 3-CNF formulas”. In: *Genetic and Evolutionary Computation Conference, GECCO 2017*. ACM, 2017, pp. 1343–1350.

[12] Maxim Buzdalov, Benjamin Doerr, and Mikhail Kever. “The unrestricted black-box complexity of jump functions”. In: *Evolutionary Computation* (2016), pp. 719–744.

[13] Tianshi Chen et al. “A New Approach for Analyzing Average Time Complexity of Population-Based Evolutionary Algorithms on Unimodal Problems”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B* 39 (2009), pp. 1092–1106.

[14] Dogan Corus, Pietro S. Oliveto, and Donya Yazdani. “Artificial Immune Systems Can Find Arbitrarily Good Approximations for the NP-Hard Partition Problem”. In: *Parallel Problem Solving from Nature, PPSN XV, Part II*. Springer, 2018, pp. 16–28.

[15] Dogan Corus and Pietro Simone Oliveto. “On the benefits of populations for the exploitation speed of standard steady-state genetic algorithms”. In: *Genetic and Evolutionary Computation Conference, GECCO 2019*. ACM, 2019, pp. 1452–1460.

[16] Dogan Corus, Pietro Simone Oliveto, and Donya Yazdani. “On the runtime analysis of the Opt-IA artificial immune system”. In: *Genetic and Evolutionary Computation Conference, GECCO 2017*. ACM, 2017, pp. 83–90.

[17] Dogan Corus et al. “Level-Based Analysis of Genetic Algorithms and Other Search Processes”. In: *IEEE Transactions on Evolutionary Computation* 22 (2018), pp. 707–719.

[18] Duc-Cuong Dang and Per Kristian Lehre. “Runtime Analysis of Non-elitist Populations: From Classical Optimisation to Partial Information”. In: *Algorithmica* 75 (2016), pp. 428–461.

[19] Duc-Cuong Dang et al. “Emergence of Diversity and Its Benefits for Crossover in Genetic Algorithms”. In: *Parallel Problem Solving from Nature, PPSN XIV*. Springer, 2016, pp. 890–900.

[20] Duc-Cuong Dang et al. “Escaping Local Optima Using Crossover With Emergent Diversity”. In: *IEEE Transactions on Evolutionary Computation* 22 (2018), pp. 484–497.

[21] Duc-Cuong Dang et al. “Escaping local optima with diversity mechanisms and crossover”. In: *Genetic and Evolutionary Computation Conference, GECCO 2016*. ACM, 2016, pp. 645–652.

[22] C.S. Davis. “Rational approximations to *e*”. In: *Journal of the Australian Mathematical Society* 25 (1978), pp. 497–502.

[23] Benjamin Doerr. “A tight runtime analysis for the cGA on jump functions: EDAs can cross fitness valleys at no extra cost”. In: *Genetic and Evolutionary Computation Conference, GECCO 2019*. ACM, 2019, pp. 1488–1496.

[24] Benjamin Doerr. “An Elementary Analysis of the Probability That a Binomial Random Variable Exceeds Its Expectation”. In: *Statistics and Probability Letters* 139 (2018), pp. 67–74.

[25] Benjamin Doerr. “An exponential lower bound for the runtime of the compact genetic algorithm on jump functions”. In: *Foundations of Genetic Algorithms, FOGA 2019*. ACM, 2019, pp. 25–33.

[26] Benjamin Doerr. “Analyzing randomized search heuristics via stochastic domination”. In: *Theoretical Computer Science* 773 (2019), pp. 115–137.

[27] Benjamin Doerr. “Better Runtime Guarantees Via Stochastic Domination”. In: *Evolutionary Computation in Combinatorial Optimization, EvoCOP 2018*. Springer, 2018, pp. 1–17.

[28] Benjamin Doerr. “Better Runtime Guarantees Via Stochastic Domination”. In: *CoRR* abs/1801.04487 (2018). arXiv: 1801.04487.

[29] Benjamin Doerr. “Does comma selection help to cope with local optima?” In: *Genetic and Evolutionary Computation Conference, GECCO 2020*. ACM, 2020, pp. 1304–1313.

[30] Benjamin Doerr. “Probabilistic Tools for the Analysis of Randomized Optimization Heuristics”. In: *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*. Springer, 2020, pp. 1–87.

[31] Benjamin Doerr and Carola Doerr. “Optimal Static and Self-Adjusting Parameter Choices for the $(1+(\lambda, \lambda))$ Genetic Algorithm”. In: *Algorithmica* 80 (2018), pp. 1658–1709.

[32] Benjamin Doerr and Carola Doerr. “Theory of parameter control for discrete black-box optimization: provable performance gains through dynamic parameter choices”. In: *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*. Ed. by Benjamin Doerr and Frank Neumann. Also available at https://arxiv.org/abs/1804.05650. Springer, 2020, pp. 271–321.

[33] Benjamin Doerr, Carola Doerr, and Franziska Ebel. “From black-box complexity to designing new genetic algorithms”. In: *Theoretical Computer Science* 567 (2015), pp. 87–104.

[34] Benjamin Doerr, Carola Doerr, and Franziska Ebel. “Lessons From the Black-Box: Fast Crossover-Based Genetic Algorithms”. In: *Genetic and Evolutionary Computation Conference, GECCO 2013*. ACM, 2013, pp. 781–788.

[35] Benjamin Doerr, Carola Doerr, and Timo Kötzing. “The unbiased black-box complexity of partition is polynomial”. In: *Artificial Intelligence* 216 (2014), pp. 275–286.

[36] Benjamin Doerr, Carola Doerr, and Timo Kötzing. “Unbiased Black-Box Complexities of Jump Functions”. In: *Evolutionary Computation* 23 (2015), pp. 641–670.

[37] Benjamin Doerr, Carola Doerr, and Johannes Lengler. “Self-adjusting mutation rates with provably optimal success rules”. In: *Genetic and Evolutionary Computation Conference, GECCO 2019*. ACM, 2019, pp. 1479–1487.

[38] Benjamin Doerr, Carola Doerr, and Frank Neumann. “Fast re-optimization via structural diversity”. In: *Genetic and Evolutionary Computation Conference, GECCO 2019*. ACM, 2019, pp. 233–241.

[39] Benjamin Doerr, Carola Doerr, and Jing Yang. “Optimal Parameter Choices via Precise Black-Box Analysis”. In: *Genetic and Evolutionary Computation Conference, GECCO 2016*. ACM, 2016, pp. 1123–1130.

[40] Benjamin Doerr, Carola Doerr, and Jing Yang. “Optimal Parameter Choices via Precise Black-box Analysis”. In: *Theoretical Computer Science* 801 (2020), pp. 1–34.

[41] Benjamin Doerr, Mahmoud Fouz, and Carsten Witt. “Sharp bounds by probability-generating functions and variable drift”. In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2011*. ACM, 2011, pp. 2083–2090.

[42] Benjamin Doerr, Edda Happ, and Christian Klein. “Crossover can provably be useful in evolutionary computation”. In: *Theoretical Computer Science* 425 (2012), pp. 17–33.

[43] Benjamin Doerr, Nils Hebbinghaus, and Frank Neumann. “Speeding Up Evolutionary Algorithms through Asymmetric Mutation Operators”. In: *Evolutionary Computation* 15 (2007), pp. 401–410.

[44] Benjamin Doerr, Daniel Johannsen, and Carola Winzen. “Multiplicative Drift Analysis”. In: *Algorithmica* 64 (2012), pp. 673–697.

[45] Benjamin Doerr, Daniel Johannsen, and Carola Winzen. “Multiplicative drift analysis”. In: *Genetic and Evolutionary Computation Conference, GECCO 2010*. ACM, 2010, pp. 1449–1456.

[46] Benjamin Doerr, Bojana Kodric, and Marco Voigt. “Lower Bounds for the Runtime of a Global Multi-Objective Evolutionary Algorithm”. In: *Congress on Evolutionary Computation, CEC 2013*. IEEE, 2013, pp. 432–439.

[47] Benjamin Doerr and Timo Kötzing. “Multiplicative up-drift”. In: *Genetic and Evolutionary Computation Conference, GECCO 2019*. ACM, 2019, pp. 1470–1478.

[48] Benjamin Doerr and Martin S. Krejca. “Significance-based estimation-of-distribution algorithms”. In: *Genetic and Evolutionary Computation Conference, GECCO 2018*. ACM, 2018, pp. 1483–1490.

[49] Benjamin Doerr and Marvin Künnemann. “Optimizing Linear Functions With the $(1 + \lambda)$ Evolutionary Algorithm—Different Asymptotic Runtimes for Different Instances”. In: *Theoretical Computer Science* 561 (2015), pp. 3–23.

[50] Benjamin Doerr and Marvin Künnemann. “Royal road functions and the $(1 + \lambda)$ Evolutionary Algorithm: Almost no speed-up from larger offspring populations”. In: *Congress on Evolutionary Computation, CEC 2013*. IEEE, 2013, pp. 424–431.

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

[52] Benjamin Doerr, Dirk Sudholt, and Carsten Witt. “When Do Evolutionary Algorithms Optimize Separable Functions in Parallel?” In: *Foundations of Genetic Algorithms, FOGA 2013*. ACM, 2013, pp. 48–59.

[53] Benjamin Doerr, Carsten Witt, and Jing Yang. “Runtime Analysis for Self-adaptive Mutation Rates”. In: *Genetic and Evolutionary Computation Conference, GECCO 2018*. ACM, 2018, pp. 1475–1482.

[54] Benjamin Doerr et al. “A Method to Derive Fixed Budget Results From Expected Optimisation Times”. In: *Genetic and Evolutionary Computation Conference, GECCO 2013*. ACM, 2013, pp. 1581–1588.

[55] Benjamin Doerr et al. “A rigorous view on neutrality”. In: *Congress on Evolutionary Computation, CEC 2007*. IEEE, 2007, pp. 2591–2597.

[56] Benjamin Doerr et al. “Black-box complexities of combinatorial problems”. In: *Theoretical Computer Science* 471 (2013), pp. 84–106.

[57] Benjamin Doerr et al. “Fast Genetic Algorithms”. In: *Genetic and Evolutionary Computation Conference, GECCO 2017*. ACM, 2017, pp. 777–784.

[58] Benjamin Doerr et al. “Faster black-box algorithms through higher arity operators”. In: *Foundations of Genetic Algorithms, FOGA 2011*. ACM, 2011, pp. 163–172.

[59] Benjamin Doerr et al. “Mutation Rate Matters Even when Optimizing Monotonic Functions”. In: *Evolutionary Computation* 21 (2013), pp. 1–27.

[60] Benjamin Doerr et al. “On the Runtime Analysis of Selection Hyper-Heuristics with Adaptive Learning Periods”. In: *Genetic and Evolutionary Computation Conference, GECCO 2018*. ACM, 2018, pp. 1015–1022.

[61] Benjamin Doerr et al. “The $(1 + \lambda)$ Evolutionary Algorithm with Self-Adjusting Mutation Rate”. In: *Algorithmica* 81 (2019), pp. 593–631.

[62] Carola Doerr. “Complexity theory for discrete black-box optimization heuristics”. In: *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*. Springer, 2020, pp. 133–211.

[63] Joseph Leo Doob. *Stochastic Processes*. Wiley, 1953.

[64] Stefan Droste. “Analysis of the (1 + 1) EA for a dynamically changing OneMax-variant”. In: *Congress on Evolutionary Computation, CEC 2002*. IEEE, 2002, pp. 55–60.

[65] Stefan Droste. “Analysis of the (1 + 1) EA for a noisy OneMax”. In: *Genetic and Evolutionary Computation Conference, GECCO 2004*. Springer, 2004, pp. 1088–1099.

[66] Stefan Droste. “Not all linear functions are equally difficult for the compact genetic algorithm”. In: *Genetic and Evolutionary Computation Conference, GECCO 2005*. ACM, 2005, pp. 679–686.

[67] Stefan Droste, Thomas Jansen, and Ingo Wegener. “On the Analysis of the $(1 + 1)$ Evolutionary Algorithm”. In: *Theoretical Computer Science* 276 (2002), pp. 51–81.

[68] Paul Erdős and Alfréd Rényi. “On Two problems of Information Theory”. In: *Magyar Tudományos Akadémia Matematikai Kutató Intézet Közleményei* 8 (1963), pp. 229–243.

[69] Tobias Friedrich, Nils Hebbinghaus, and Frank Neumann. “Comparison of simple diversity mechanisms on plateau functions”. In: *Theoretical Computer Science* 410 (2009), pp. 2455–2462.

[70] Tobias Friedrich, Nils Hebbinghaus, and Frank Neumann. “Plateaus Can Be Harder in Multi-objective Optimization”. In: *Theoretical Computer Science* 411 (2010), pp. 854–864.

[71] Tobias Friedrich et al. “Fast Building Block Assembly by Majority Vote Crossover”. In: *Genetic and Evolutionary Computation Conference, GECCO 2016*. ACM, 2016, pp. 661–668.

[72] Josselin Garnier, Leila Kallel, and Marc Schoenauer. “Rigorous Hitting Times for Binary Mutations”. In: *Evolutionary Computation* 7 (1999), pp. 173–203.

[73] Oliver Giel and Ingo Wegener. “Evolutionary Algorithms and the Maximum Matching Problem”. In: *Symposium on Theoretical Aspects of Computer Science, STACS 2003*. Springer, 2003, pp. 415–426.

[74] Christian Gießen and Timo Kötzing. “Robustness of Populations in Stochastic Environments”. In: *Algorithmica* 75 (2016), pp. 462–489.

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

[76] Brian W. Goldman and William F. Punch. “Parameter-less population pyramid”. In: *Genetic and Evolutionary Computation Conference, GECCO 2014*. ACM, 2014, pp. 785–792.

[77] Spencer Greenberg and Mehryar Mohri. “Tight Lower Bound on the Probability of a Binomial Exceeding its Expectation”. In: *Statistics and Probability Letters* 86 (2014), pp. 91–98.

[78] Bruce Hajek. “Hitting-time and occupation-time bounds implied by drift analysis with applications”. In: *Advances in Applied Probability* 14 (1982), pp. 502–525.

[79] Václav Hasenöhrl and Andrew M. Sutton. “On the runtime dynamics of the compact genetic algorithm on jump functions”. In: *Genetic and Evolutionary Computation Conference, GECCO 2018*. ACM, 2018, pp. 967–974.

[80] Jun He and Xin Yao. “A study of drift analysis for estimating computation time of evolutionary algorithms”. In: *Natural Computing* 3 (2004), pp. 21–35.

[81] Jun He and Xin Yao. “Drift Analysis and Average Time Complexity of Evolutionary Algorithms”. In: *Artificial Intelligence* 127 (2001), pp. 57–85.

[82] Hsien-Kuei Hwang and Carsten Witt. “Sharp bounds on the runtime of the $(1+1)$ EA via drift analysis and analytic combinatorial tools”. In: *Foundations of Genetic Algorithms, FOGA 2019*. ACM, 2019, pp. 1–12.

[83] Hsien-Kuei Hwang et al. “Probabilistic analysis of the $(1+1)$-evolutionary algorithm”. In: *Evolutionary Computation* 26 (2018), pp. 299–345.

[84] Jens Jägersküpper and Tobias Storch. “When the Plus Strategy Outperforms the Comma Strategy and When Not”. In: *Foundations of Computational Intelligence, FOCI 2007*. IEEE, 2007, pp. 25–32.

[85] Jens Jägersküpper and Carsten Witt. “Rigorous Runtime Analysis of a $(\mu+1)$ ES for the Sphere Function”. In: *Genetic and Evolutionary Computation Conference, GECCO 2005*. ACM. 2005, pp. 849–856.

[86] Thomas Jansen. *Analyzing Evolutionary Algorithms – The Computer Science Perspective*. Springer, 2013. ISBN: 978-3-642-17338-7.

[87] Thomas Jansen, Kenneth A. De Jong, and Ingo Wegener. “On the Choice of the Offspring Population Size in Evolutionary Algorithms”. In: *Evolutionary Computation* 13 (2005), pp. 413–440.

[88] Thomas Jansen and Ingo Wegener. “Evolutionary algorithms - how to cope with plateaus of constant fitness and when to reject strings of the same fitness”. In: *IEEE Trans. Evolutionary Computation* 5 (2001), pp. 589–599.

[89] Thomas Jansen and Ingo Wegener. “Real royal road functions–where crossover provably is essential”. In: *Discrete Applied Mathematics* 149 (2005), pp. 111–125.

[90] Thomas Jansen and Ingo Wegener. “The analysis of evolutionary algorithms—a proof that crossover really can help”. In: *Algorithmica* 34 (2002), pp. 47–66.

[91] Thomas Jansen and Christine Zarges. “On benefits and drawbacks of aging strategies for randomized search heuristics”. In: *Theoretical Computer Science* 412 (2011), pp. 543–559.

[92] Timo Kötzing. “Concentration of First Hitting Times Under Additive Drift”. In: *Algorithmica* 75 (2016), pp. 490–506.

[93] Jérémie Labroquère et al. “Evolutionary Constrained Optimization for a Jupiter Capture”. In: *Parallel Problem Solving from Nature, PPSN 2014*. Springer, 2014, pp. 262–271.

[94] Jörg Lässig and Dirk Sudholt. “Design and analysis of migration in parallel evolutionary algorithms”. In: *Soft Computing* 17 (2013), pp. 1121–1144.

[95] Per Kristian Lehre. “Fitness-levels for non-elitist populations”. In: *Genetic and Evolutionary Computation Conference, GECCO 2011*. ACM, 2011, pp. 2075–2082.

[96] Per Kristian Lehre. “Negative Drift in Populations”. In: *Parallel Problem Solving from Nature, PPSN 2010*. Springer, 2010, pp. 244–253.

[97] Per Kristian Lehre and Carsten Witt. “Black-Box Search by Unbiased Variation”. In: *Algorithmica* 64 (2012), pp. 623–642.

[98] Per Kristian Lehre and Carsten Witt. “Concentrated Hitting Times of Randomized Search Heuristics with Variable Drift”. In: *Proc. of the 25th International Symposium on Algorithms and Computation (ISAAC 2014)*. Vol. 8889. Lecture Notes in Computer Science. Springer, 2014, pp. 686–697.

[99] Per Kristian Lehre and Xin Yao. “On the Impact of Mutation-Selection Balance on the Runtime of Evolutionary Algorithms”. In: *IEEE Transactions on Evolutionary Computation* 16 (2012), pp. 225–241.

[100] Johannes Lengler. “A General Dichotomy of Evolutionary Algorithms on Monotone Functions”. In: *Parallel Problem Solving from Nature, PPSN 2018, Part II*. Springer, 2018, pp. 3–15.

[101] Johannes Lengler. “Drift Analysis”. In: *CoRR* abs/1712.00964 (2017). arXiv: 1712.00964.

[102] Andrei Lissovoi, Pietro Simone Oliveto, and John Alasdair Warwicker. “On the runtime analysis of generalised selection hyper-heuristics for pseudo-Boolean optimisation”. In: *Genetic and Evolutionary Computation Conference, GECCO 2017*. ACM, 2017, pp. 849–856.

[103] Elisabeth Lübbecke, Marco E. Lübbecke, and Rolf H. Möhring. “Ship Traffic Optimization for the Kiel Canal”. In: *Operations Research* 67 (2019), pp. 791–812.

[104] Carl D. Meyer, ed. *Matrix Analysis and Applied Linear Algebra*. Society for Industrial and Applied Mathematics, 2000, pp. 661–704.

[105] Vladimir Mironovich and Maxim Buzdalov. “Evaluation of heavy-tailed mutation operator on maximum flow test generation problem”. In: *Genetic and Evolutionary Computation Conference, GECCO 2017, Companion Material*. ACM, 2017, pp. 1423–1426.

[106] Melanie Mitchell, Stephanie Forrest, and John H. Holland. “The Royal Road for Genetic Algorithms: Fitness Landscapes and GA Performance”. In: *Proc. of the First European Conference on Artificial Life*. MIT Press, 1992, pp. 245–254.

[107] Heinz Mühlenbein. “Evolutionary Algorithms: Theory and Applications”. In: *Local Search in Combinatorial Optimization*. Wiley, 1993.

[108] Heinz Mühlenbein. “How Genetic Algorithms Really Work: Mutation and Hillclimbing”. In: *Parallel Problem Solving from Nature, PPSN 1992*. Elsevier, 1992, pp. 15–26.

[109] Frank Neumann, Pietro Simone Oliveto, and Carsten Witt. “Theoretical analysis of fitness-proportional selection: landscapes and efficiency”. In: *Genetic and Evolutionary Computation Conference, GECCO 2009*. ACM, 2009, pp. 835–842.

[110] Frank Neumann, Dirk Sudholt, and Carsten Witt. “Analysis of different MMAS ACO algorithms on unimodal functions and plateaus”. In: *Swarm Intelligence* 3 (2009), pp. 35–68.

[111] Frank Neumann and Ingo Wegener. “Randomized Local Search, Evolutionary Algorithms, and the Minimum Spanning Tree Problem”. In: *Genetic and Evolutionary Computation Conference, GECCO 2004*. Springer, 2004, pp. 713–724.

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

[113] Pietro Simone Oliveto and Carsten Witt. “Erratum: Simplified Drift Analysis for Proving Lower Bounds in Evolutionary Computation”. In: *CoRR* abs/1211.7184 (2012). arXiv: 1211.7184.

[114] Axel de Perthuis de Laillevault, Benjamin Doerr, and Carola Doerr. “Money for Nothing: Speeding Up Evolutionary Algorithms Through Better Initialization”. In: *Genetic and Evolutionary Computation Conference, GECCO 2015*. ACM, 2015, pp. 815–822.

[115] Eduardo Carvalho Pinto and Carola Doerr. “Towards a more practice-aware runtime analysis of evolutionary algorithms”. In: *CoRR* abs/1812.00493 (2018).

[116] Chao Qian, Yang Yu, and Zhi-Hua Zhou. “A Lower Bound Analysis of Population-Based Evolutionary Algorithms for Pseudo-Boolean Functions”. In: *International Conference on Intelligent Data Engineering and Automated Learning, IDEAL 2016*. Springer. 2016, pp. 457–467.

[117] Yuval Rabani, Yuri Rabinovich, and Alistair Sinclair. “A Computational View of Population Genetics”. In: *Random Structures & Algorithms* 12 (1998), pp. 313–334.

[118] Amirhossein Rajabi and Carsten Witt. “Self-Adjusting Evolutionary Algorithms for Multimodal Optimization”. In: *CoRR* abs/2004.03266 (2020).

[119] Michael Renardy and Robert C. Rogers. *An Introduction to Partial Differential Equations*. Texts in Applied Mathematics. Springer, 2004.

[120] Herbert Robbins. “A Remark on Stirling’s Formula”. In: *The American Mathematical Monthly* 62 (1955), pp. 26–29.

[121] Jonathan E. Rowe and Dirk Sudholt. “The choice of the offspring population size in the $(1, \lambda)$ evolutionary algorithm”. In: *Theoretical Computer Science* 545 (2014), pp. 20–38.

[122] Günter Rudolph. “Convergence of Evolutionary Algorithms in General Search Spaces”. In: *International Conference on Evolutionary Computation*. IEEE, 1996, pp. 50–54.

[123] Günter Rudolph. *Convergence Properties of Evolutionary Algorithms*. Verlag Dr. Kovăc, 1997.

[124] Tobias Storch and Ingo Wegener. “Real royal road functions for constant population size”. In: *Theoretical Computer Science* 320 (2004), pp. 123–134.

[125] Dirk Sudholt. “A New Method for Lower Bounds on the Running Time of Evolutionary Algorithms”. In: *IEEE Transactions on Evolutionary Computation* 17 (2013), pp. 418–435.

[126] Dirk Sudholt. “Analysing the Robustness of Evolutionary Algorithms to Noise: Refined Runtime Bounds and an Example Where Noise is Beneficial”. In: *Algorithmica* (2020). To appear.

[127] Dirk Sudholt. “General Lower Bounds for the Running Time of Evolutionary Algorithms”. In: *Parallel Problem Solving from Nature, PPSN 2010*. Springer, 2010, pp. 124–133.

[128] Dirk Sudholt. “On the robustness of evolutionary algorithms to noise: refined results and an example where noise helps”. In: *Genetic and Evolutionary Computation Conference, GECCO 2018*. ACM, 2018, pp. 1523–1530.

[129] Dirk Sudholt. “The impact of parametrization in memetic evolutionary algorithms”. In: *Theoretical Computer Science* 410 (2009), pp. 2511–2528.

[130] Joe Suzuki. “A Markov Chain Analysis on Simple Genetic Algorithms”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 25 (1995), pp. 655–659.

[131] Olivier Teytaud and Sylvain Gelly. “General lower bounds for evolutionary algorithms”. In: *Parallel Problem Solving from Nature, PPSN 2006*. Springer, 2006, pp. 21–31.

[132] Abraham Wald. “Some generalizations of the theory of cumulative sums of random variables”. In: *The Annals of Mathematical Statistics* 16 (1945), pp. 287–293.

[133] Ingo Wegener. “Theoretical Aspects of Evolutionary Algorithms”. In: *International Colloquium on Automata, Languages, and Programming, ICALP 2001*. Springer, 2001, pp. 64–78.

[134] Darrell Whitley et al. “Exploration and Exploitation Without Mutation: solving the Jump Function in Θ(n) Time”. In: *Parallel Problem Solving from Nature, PPSN 2018, Part II*. Springer, 2018, pp. 55–66.

[135] Carsten Witt. “Fitness levels with tail bounds for the analysis of randomized search heuristics”. In: *Information Processing Letters* 114 (2014), pp. 38–41.

[136] Carsten Witt. “Population size versus runtime of a simple evolutionary algorithm”. In: *Theoretical Computer Science* 403 (2008), pp. 104–120.

[137] Carsten Witt. “Population Size vs. Runtime of a Simple EA”. In: *Congress on Evolutionary Computation, CEC 2003*. IEEE. 2003, pp. 1996–2003.

[138] Carsten Witt. “Runtime Analysis of the (μ + 1) EA on Simple Pseudo-Boolean Functions”. In: *Evolutionary Computation* 14 (2006), pp. 65–86.

[139] Carsten Witt. “Tight Bounds on the Optimization Time of a Randomized Search Heuristic on Linear Functions”. In: *Combinatorics, Probability and Computing* 22 (2013), pp. 294–318.

[140] Yang Yu, Chao Qian, and Zhi-Hua Zhou. “Switch Analysis for Running Time Analysis of Evolutionary Algorithms”. In: *IEEE Transactions on Evolutionary Computation* 19 (2015), pp. 777–792.

List of Figures

List of Tables