Benefits of Sexual Reproduction in Evolutionary Computation

Dissertation vorgelegt von
Diplom-Informatikerin
Madeleine Friedrich geborene Theile
aus Lutherstadt Wittenberg

Von der Fakultät II – Mathematik und Naturwissenschaften
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
– Dr. rer. nat.–

genehmigte Dissertation

Promotionsausschuss:
Vorsitzender: Prof. Dr. Dietmar Hömberg
Gutachter: Prof. Dr. Martin Skutella
Gutachter: Prof. Dr. Frank Neumann

Tag der wissenschaftlichen Aussprache: 28. Juni 2013

Berlin 2013
D 83

Zusammenfassung

Evolutionäre Algorithmen werden aufgrund ihrer leichten Implementierbarkeit und vielfältigen Anpassungsmöglichkeiten häufig in der industriellen Praxis eingesetzt. Sie sind hoch-erfolgreich bei der Lösung praktischer kombinatorischer Optimierungsprobleme, welche zu komplex sind für klassische, algorithmische Ansätze. Jedoch steckt unser Verständnis der darunter liegenden stochastischen Prozesse im Gegensatz zu ihrer praktischen Bedeutung noch in den Kinderschuhen. Der Entwicklung evolutionärer Algorithmen liegen biologische Konzepte wie die Fortpflanzung zugrunde. So scheint es in der Natur für höhere Lebewesen ein inhärenter Vorteil zu sein, sich sexuell fortzupflanzen. Diese Beobachtung nutzen viele erfolgreiche evolutionäre Algorithmen in der Praxis durch den Einsatz von sogenannten Rekombinationsstrategien. Jedoch gilt auch für unser theoretisches Verständnis von Rekombination, dass dieses noch in den Anfängen steckt. Diese Fragestellung ist Grundlage der vorliegenden Arbeit: Es wird das Verhalten verschiedener evolutionären Algorithmen mit Hilfe mathematischer Methoden untersucht – der Fokus liegt dabei auf den Vorteilen von sexueller Fortpflanzung.

Zwei klassische Optimierungsprobleme sind das NP-schwere Problem des Handlungsreisenden (TSP) und das in Polynomialzeit lösbare Problem der Berechnung aller Paare von kürzesten Wegen (APSP). Beide sind Vertreter der großen Klasse von kombinatorischen Programmierungsproblemen, die durch ein dynamisches Programm beschrieben sind. In Kapitel 2 dieser Arbeit beschäftigen wir uns mit der mathematischen Beschreibung der gemeinsamen Eigenschaften von Problemen aus dieser Klasse und leiten eine generische Repräsentation her. Es zeigt sich, dass dies einen prototypischen evolutionären Algorithmus (ohne Rekombination) ermöglicht, der Lösungen in der Art eines dynamischen Programmes konstruiert, ohne die eigentliche Struktur des Problems zu kennen. Hierfür beweisen wir obere Schranken für die Laufzeit dieses Algorithmus. Bemerkenswert dabei ist, dass dies das zweite Mal überhaupt ist, dass Laufzeitschranken eines evolutionären Algorithmus für eine große Klasse von Problemen gezeigt werden können. Experimente für TSP zeigen empirisch, dass die Verwendung von Rekombination zu einer zusätzlichen Beschleunigung führt.

Genetische Algorithmen sind evolutionäre Algorithmen, welche sexuelle Fortpflanzung in Form eines Rekombinationsoperators nutzen. Um unser Verständnis wichtiger Prinzipien von Rekombination zu verbessern, wird in Kapitel 3 die Laufzeit genetischer Algorithmen auf bekannten künstlichen pseudo-Booleschen Funktionen analysiert. Theoretische und empirische Resultate zeigen einen substantielle Beschleunigung für sogenannte “functions of unitati-

on" bei Verwendung eines Fitness-invarianten Bit-Shuffling-Operators. Wir betrachten zusätzlich einfache, genetische Algorithmen ohne Shuffling und untersuchen das Zusammenspiel von Mutation und Rekombination auf der Testfunktion Jump. Für kleine Rekombinationsraten beweisen wir, dass Mutationen auch für sehr kleine Populationen ausreichend Diversität produziert, um die Testfunktion zu optimieren. Im Gegensatz dazu, verringert eine große Rekombinationsrate die Diversität schneller als Mutation sie erzeugen kann. Beide Effekte haben einen signifikanten Einfluss auf die Optimierungszeit. Wir vervollständigen unsere theoretischen Untersuchungen durch Monte-Carlo-Simulationen.

APSP ist das bisher einzige kombinatorische Optimierungsproblem, für welches der Nutzen von Rekombination mathematisch bewiesen werden konnte. In Kapitel 4 verbessern wir die besten bekannten Laufzeitschranken für APSP und beweisen, dass unsere neuen Schranken asymptotisch optimal sind. Unsere neuartige, rigorose Analyse ermöglicht neue Einsichten, wie Rekombination in noch unbekannte Gegenden des Suchraums vorstößt, und es komplementär dazu dem Mutationsoperator überlässt, die nähere Nachbarschaft zu untersuchen. In der Praxis nutzen genetische Algorithmen oft verfeinerte Rekombinationsstrategien, welche beispielsweise ungültige Nachkommen reparieren. Für zwei derartige Konzepte beweisen wir für APSP eine kleinere Optimierungszeit. Wir erweitern diese Ansätze für die NP-schwere mehrkriterielle Variante von APSP und präsentieren einen genetischen Algorithmus mit einem speziellen Diversitätsmaß, welches die Größe der Population vom Benutzer einstellbar steuert. Dies ist notwendig, da die Pareto-Front im worst-case exponentiell groß werden kann. Wir beweisen, dass der resultierende genetische Algorithmus ein voll polynomielles randomisiertes Approximationsschema (FPRAS) ist. Dies zeigt, dass Rekombination zu einer besseren Laufzeit führen kann als die besten zuvor bekannten Resultate. Mit dieser Analyse wurde zum ersten Mal mathematisch gezeigt, dass Rekombination für ein NP-schweres Optimierungsproblem vorteilhaft ist.

Abstract

Evolutionary algorithms have proven to be highly successful for real world combinatorial optimization problems too complex to be handled by traditional algorithmic approaches. In stark contrast to their practical relevance, we only start to theoretically understand the stochastic processes which govern these optimization algorithms. The development of evolutionary algorithms was inspired by biological observations. In nature there seems to be an inherent advantage of sexual recombination compared to mere asexual reproduction. Thus many successful evolutionary algorithms in practice make use of recombination strategies. However, our theoretical understanding of crossover in evolutionary computation is very limited. This thesis studies the behavior of several evolutionary algorithms with a focus on the benefits of sexual reproduction.

Two archetypical combinatorial optimization problems are the NP-hard Traveling Salesperson Problem (TSP) and the polynomial-time solvable all-pairs shortest path problem (APSP). Both are representatives of the large class of combinatorial optimization problems with a dynamic programming structure. Chapter 2 of this thesis presents a mathematical description of the common properties of this class and derives a generic representation. This enables a prototypical evolutionary algorithm (without crossover) to construct solutions in a dynamic programming fashion without access to the structure of the problem. We prove upper bounds on the running time of this algorithm. This is the second time ever that runtime bounds of evolutionary algorithms for a large class of problems could be proven. Empirical experiments on TSP show that the use of crossover can lead to additional speed-ups.

Genetic algorithms are evolutionary algorithms which employ sexual reproduction in form of some crossover operators. In order to get a better understanding of the working principles of crossover, Chapter 3 analyzes the runtime of genetic algorithms on common artificial pseudo-Boolean functions. Theoretical and empirical results show substantial speedups for functions of unitation when combined with a fitness-invariant bit shuffling operator. We also consider a simple genetic algorithm without shuffling and investigate the interplay of mutation and crossover on the test function Jump. We prove for small crossover probabilities that subsequent mutations create sufficient diversity, even for very small populations. Contrarily, with high crossover probabilities crossover tends to lose diversity more quickly than mutation can create it. Both effects have a drastic impact on the performance on Jump. We complement our theoretical findings by Monte Carlo simulations.

APSP is the only combinatorial optimization problem for which the usefulness of crossover could be rigorously shown. In Chapter 4 we improve the best known runtime for APSP and prove that our bounds are asymptotically optimal. Our analysis shows how crossover can be analyzed with rigorous methods. This gives new insights on how crossover explores the search space at large, leaving the exploration of the closer neighborhood to the mutation operator. Real-world genetic algorithms often use different and more refined recombination strategies, which for example repair infeasible offsprings. For two such concepts we prove a smaller optimization time. We also extend these approaches to the NP-hard multi-criteria variant of APSP. We present a genetic algorithm with a diversity mechanism that lets the user control the size of the population. This is necessary as otherwise the size of the Pareto front can become exponential. We prove that the resulting genetic algorithm is a fully-polynomial time randomized approximation scheme (FPRAS). This shows that crossover leads to better worst case bounds than previous known results. This is the first time that rigorous runtime analyses have shown the usefulness of crossover for an NP-hard optimization problem.

Acknowledgments

Many people have supported me during my work on this thesis. I am most indebted to Martin Skutella for offering me the chance to obtain my PhD in his vibrant group. I admire his ability to hold up a spirit of support for everyone. He granted me the freedom to pursue my own research interests. Thanks, Martin, I could not have imagined to be anywhere else than in your group.

Thanks to the organizers of the Theory of Evolutionary Algorithms seminar at Schloss Dagstuhl (2011). I was honored to become part of this inspiring surroundings. I took home several inspiring thoughts resulting in a best paper award.

I would also like to express my gratitude to all of my co-authors. My work has greatly benefited from talking and discussing with you. Thank you for this!

I thank Martin Skutella and Frank Neumann for their support of my enlightening and fruitful research stay in Adelaide in 2011. My special thanks go to my office mate Jannik Matuschke who endured long hours on Skype discussing multi-commodity network flows with me.

Wolfgang Welz supported me in conducting TSP experiments. I very much appreciate his patience in reviving those results several years later.

Thanks to all those people from COGA whom I could not name here. You made my time at TU Berlin as pleasurable as possibly possible.

Contents

1 Introduction

Classical combinatorial optimization problems are in general very well understood. Either they possess a classical algorithm for their solution like shortest paths, flows, and matchings, or there exists a well-developed machinery of (mixed-integer) linear programs for their (heuristic) solutions. Unfortunately, many real world optimization problems withstand a solution with these traditional algorithmic approaches because (i) they do not posses a (linear) function that describes them, (ii) the constraints given for the problem are neither linear nor quadratic, or (iii) there is no (closed-form) function available at all. In these cases there is a clear trade-off between the time, cost and knowledge to invest in the development of an algorithm on the one hand, and its efficiency on the other hand.

Black-Box Optimization. If the only access to a problem is via some arbitrary quality measure, then this is known as *black-box optimization*. In this setting, all possible solutions are contained in some search space $\mathcal{S}$. Their respective quality is measured by a function $f\colon \mathcal{S} \to \mathbb{R}^d$, which is considered to be a black box. The function f is not known algebraically; it might be determined by an experiment or some simulations. This implies that in contrast to classical optimization problems, we do not know any derivatives or Hessian matrix of the function f. The case $d = 1$ is called single-objective black-box optimization, where we seek a solution $x^* \in \mathcal{S}$ that maximizes (or minimizes) the value of $f(x^*)$. In case of multiple objectives ($d > 1$) which are optimized at the same time (like time and cost), we seek a set of mutually non-dominating solutions called Pareto-front (see Section 4.7 for details). A black-box optimization algorithm optimizes such a function f by iteratively sampling $x \in \mathcal{S}$ and evaluating $f(x)$. The optimization time is then the number of evaluations of the function f.

A black box optimization algorithm is a problem-independent algorithm with standardized components which aims at optimizing fitness functions defined on some common search space $\mathcal{S}$. It is allowed to adjust its components to the considered problem. The design of a black box optimization algorithm should only involve the following three steps, which makes it very easy to apply such methods to a newly given problem:

1. Choose a representation of possible solutions.
2. Determine a function to evaluate the quality of a solution.
3. Define operators for the variation & selection of new solutions.

Stochastic search algorithms are black box optimization algorithms and as such have proven to be highly successful for real world combinatorial optimization problems that cannot be handled by traditional algorithmic approaches [142]. They are a premier choice for complex optimization problems that are highly non-linear, dynamic, and/or stochastic. They have accomplished many practical successes [53, 70, 104, 105, 136, 168]; companies like SolveIT Software[1], which exclusively develop stochastic search algorithms, generate an eight-digit euro sales figure each year. While these algorithms work well for many optimization problems in practice, a satisfying and rigorous mathematical understanding of their performance is still highly missing. This is mainly due to the fact that stochastic search algorithms make use of random decisions in different components. This leads to stochastic processes that are notoriously hard to analyze [3, 126]. Rabani, Rabinovich, and Sinclair [124] for example classify genetic algorithms as quadratic, hence non-linear systems which "are usually impossible to analyze". Another main advantage of stochastic search algorithms is that they are easy to parallelize [149] and viewing them in the light of a tremendous number of increasing processors on multi-core computers, one can expect that the number of applications to interesting real-world problems [10, 78, 156, 157, 158] will get a further boost during the next decade.

Runtime Analysis. Runtime analysis has emerged as a mathematical tool to explain their practical success in theory (see the recent books of Neumann and Witt [113], Auger and Doerr [4], and Jansen [81] for the current state of the art). In contrast to early research on models with e. g. infinite population sizes and studies on the convergence in the limit, modern runtime analysis rather analyzes the algorithm itself in a rigorous mathematical fashion. On the other hand, it is much less ambitious in terms of generality as one considers specific algorithms on specific problems or problem classes. In this way, it

[1] `http://www.solveitsoftware.com/`

allows to estimate the performance of stochastic search algorithms in a very precise way. Bounds on the running time are obtained until a desirable solution such as a global optimum has been found. This time is called *optimization time*. The resulting bounds apply to problems of growing size and hence allow to assess their scalability. Our aim is to understand the conditions under which these algorithms prove to be successful. In the long run it is the task of a theoretician to give advise to practitioners on how to design a stochastic search algorithm for their specific problem.

Some Disclaimer. We want to point out that we do nature-inspired optimization. This is unrelated to bioinformatics which goes the other way around and uses tools from computer science to solve problems in biology. Besides this, there is also a recent trend to explain biological phenomenon with methods from (theoretical) computer science. Examples for this are mixability in evolution theory [98, 99], evolvability in computational learning [153], convergence of bird flocks [18, 19], shortest path computation of slime molds [12], and maximal independent set computation in fly's nervous system [1]. The concept which comes closest to our work is mixability [98, 99]. We discuss this on page 9.

However, in nature-inspired optimization we incorporate principles known to be successful in nature into computer science to solve optimization problems. This is the case for evolutionary algorithms (EAs) which are a subclass of stochastic search algorithms.

The Evolutionary Cycle. Our discussion of the following description of evolutionary algorithms follows the notation of the recent book of Blum et al. [11]. According to them evolutionary algorithms originally were all kind of stochastic search algorithms mimicking the evolution of nature including Genetic Algorithms, Genetic Programming, Evolution Strategies and Evolutionary Programming. On the other hand EAs can also be understood as all kind of "population-based algorithms with a notion of randomness and selection". According to personal communication with Prof. Zbigniew Michalewicz, author of [11], he deems it hard "to establish semantic border between the algorithms".

An evolutionary algorithm progressively improves a solution in search for the optimum. Its purpose is to solve a general class of problems without any knowledge of the inner structure of the problem. Thus, it treats the objective function f as a "black-box" and gains knowledge about the problem by assessing formerly evaluated candidate solutions only. This knowledge is used to construct a new and hopefully better candidate solution.

In general, an evolutionary algorithm includes

(a) individuals from one or more populations that compete for some limited resources,
(b) a notion of fitness that allows an individual to survive and reproduce,
(c) a notion of variation in reproduction, offspring resemble their parents but are not alike.

The Darwinian principles underlying evolution in nature suggest that species improve their fitness over generations. This typically means that they adapt to a changing environment. A typical population-based EA consists of the following elementary cycle shown in Figure 1.1.

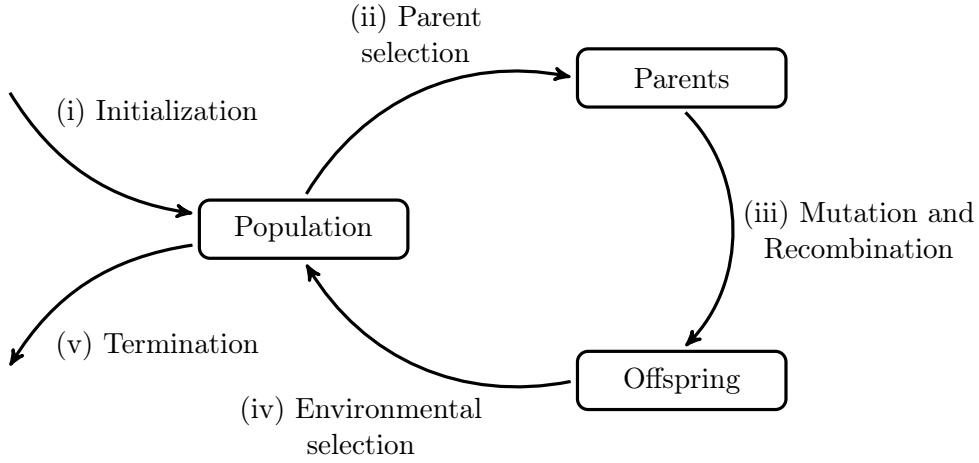
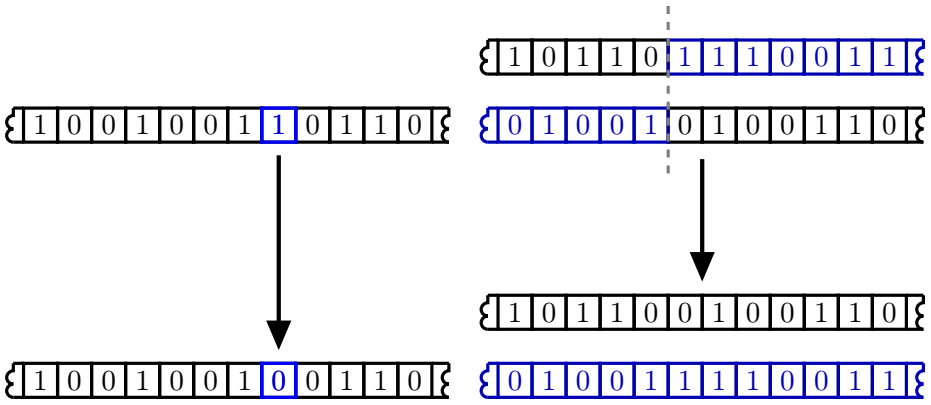


Figure 1.1: The basic cycle of EAs.

During the initialization (i) the initial population consisting of $n > 0$ individuals is created either according to some probability distribution or with seeds of known solutions. The subsequent parent selection (ii) lets individuals enter the reproduction pool according to some probability distribution. This can either be a selection in accordance with the fitness of an individual or panmictic selection. This thesis uses only the latter mechanism, which means, that parents are chosen uniformly at random from the population. During the reproduction phase (iii) offspring are derived from selected individuals based on some variation operations (mutation and recombination). Mutation applies some local changes to a single individual. Recombination, also called crossover, takes (at least) two individuals to recombine their information into an offspring, compare Figure 1.2 for bitstring representations. It depends on the choice of the algorithm designer as well as the desired size of the population in which way the offspring enters the population during the environmental selection step (iv). The two choices are plus- and comma-selection, and



(a) One-bit Mutation (b) One-point Crossover

Figure 1.2: Illustration of variation operators on bit strings $\{0,1\}^n$.

both variants involve the evaluation of an individuals objective value, the so-called fitness. In comma-selection, e. g. for a (μ, λ) EA, the parent population has size μ and the next generation is chosen from among λ offspring, where typically $\lambda \gg \mu$. This especially means, that the individuals in the parent-generation are not involved in the selection process and die. In plus-selection, e. g. for a $(\mu + \lambda)$ EA, the next generation is selected from the union of the μ parents and the λ offspring created by some variation operators. In this thesis we will exclusively use the so-called elitist selection, a plus-selection scheme, which selects the population among the best parents and offspring. An individual is only allowed to enter the population if its fitness is not worse than a comparable individual from the parent-generation. In particular, it never looses the best solution seen so far. An additional diversity measure may be part of the environmental selection to prevent comparison with the wrong individuals. If, finally, the termination criterion (v) is met which can be a predefined iteration bound or a quality measure the evolutionary process stops, otherwise it continues with step (ii).

The *nomenclature* in this thesis treats the algorithms that use *mutation only* as *evolutionary algorithms (EA)*, e. g. $(\mu + 1)$ EA, and algorithms *with crossover* as *genetic algorithms (GA)*, e. g. $(\mu + 1)$ GA. We will use *evolutionary computation (EC)* as generic term for both. We remark that there are other stochastic search algorithms like Ant Colony Optimization [47] or Particle Swarm Optimization [87], which are beyond the scope of this thesis and are not discussed.

1.1 Evolutionary Algorithms

In the 80's there were high hopes set into evolutionary computation. Practitioners were hoping that this class of new algorithms might perform better than any other algorithm. These hopes were dashed when Wolpert and Macready [166] published their No-Free-Lunch Theorem (NFL) in 1997. This by now classical theorem states that if the class $F \subseteq \{f : \mathcal{S} \to R\}$ for sets $\mathcal{S}$ (finite search space) and R (finite domain) is closed under permutation, then each two stochastic search algorithms that never evaluate a search point twice perform the same, averaged over all functions of F. Taken to the extreme, no stochastic search algorithm on average performs better than blind random search. Droste, Jansen, and Wegener [50] showed that this even holds for functions that are interesting in practice. For every function that is efficiently optimized by a specific stochastic search algorithms we find many other functions with a comparable size of representation on which the algorithm performs poorly.

The No-Free-Lunch Theorem makes a statement for all stochastic search heuristics on all functions. However, this still leaves hope that our favorite search algorithm performs good on many relevant function classes. This should make clear that an analysis of these algorithms with respect to their runtime makes sense only for specific classes of functions or specific classes of problems.

Pseudo-Boolean Optimization. This line of research started with results on simple pseudo-Boolean functions. Early results cope with convergence properties [134] of Markov chains that model the behavior of simple EAs. Droste, Jansen, and Wegener [49] prove upper and lower bounds on the class of linear and unimodal functions, Jansen and Wegener [84] cope with plateaus of constant fitness. Numerous other test function have been developed to demonstrate a specific ability of an evolutionary algorithm, e. g. [49, 161]. Together with the research on the mutation operator [33, 35, 83], different selection and diversity maintaining mechanism [63, 65, 79, 80, 115], nowadays, we have a fairly good understanding of the dynamics in Pseudo-Boolean optimization without populations. There is even some early research on how EAs cope with a dynamically changing environment [48, 82, 132]. However, these investigations typically concentrate on the (1+1) EA which can be analyzed as a Markov-chain as opposed to a population-based EA in which stochastic dependencies between individuals play a role. However, a practitioner would always use the advantages of a population when solving a problem. A population offers the chance to store a large variety of solutions with different properties that are beneficial for the optimization process.

Populations are obviously necessary if we consider parallel as well as multi-objective settings. Here, research has just been started on how to parallelize EAs to keep pace with the progress of modern architectures [92]. Though populations are of course crucial for multi-objective optimization problems [94], only little success so far has been achieved for a deeper understanding of the general dynamics in them [95, 96, 143, 163]. Apart from some approaches to solve combinatorial optimization problems with a multi-objective setting [64, 110] (where the additional objectives serve to incorporate constraints or create additional diversity [17]), research on multi-objective optimization mainly concentrates on convergence [93] and diversity [76] by indicators like the hypervolume indicator, with which the quality between different sets can be compared, cf. e. g. [9, 13, 14, 16, 116]

Combinatorial Optimization. The NFL shows the basic limitations of stochastic search algorithms and hence evolutionary computation. In applications, it is hardly ever realistic that nothing is known about the problem at hand. There is almost always some knowledge about typical solutions. Incorporating problem-specific knowledge can be crucial for the success and the efficiency of evolutionary algorithms. One of the important issues when designing successful evolutionary algorithms is to choose a suitable representation of possible solutions together with good variation operators. Different representations and variation operators have been discussed for combinatorial optimization problems (see e. g. [89, 127, 133]). Interesting results on the runtime behavior of evolutionary algorithms have been obtained for a wide range of combinatorial optimization problems (see [4, 113] for a comprehensive presentation). This includes well-known problems such as sorting and shortest paths [138], spanning trees [112] and matroids [128], Eulerian cycles [36, 109], maximum matchings [68], and minimum cuts [66, 110, 114]. There are also some results on evolutionary algorithms acting as approximation algorithms for NP-hard problems like partition [164], covering [64], and multi-objective shortest path [75, 111] problems. Recently, it has been observed that evolutionary algorithms even carry the capability to act as fixed-parameter tractable algorithms (see [59, 62, 117] for books on parameterized complexity) for NP-hard problems [91, 147, 148].

The goal of all the aforementioned studies is to increase the understanding of evolutionary algorithms in a rigorous way and provide guidelines for the application of these methods. One cannot expect to beat the best known problem-specific algorithms with black-box algorithms. With such studies we want to gain new insights on how evolutionary algorithms behave on natural optimization problems and give insights into the important components that make such algorithms successful.

1.2 Genetic Algorithms

The interest in genetic algorithms introduced by Holland [73] certainly stems from the biological fact that all higher organisms reproduce sexually. Thus, there seems to be a strong evolutionary advantage of sexual reproduction which in evolutionary computation is mimicked by the use of crossover operators. However, it is still debated how, when and why crossover is beneficial. All of the results discussed in the last section have in common that they are based on evolutionary algorithms (using mutation only). Understanding the impact of crossover on performance is still a major problem in the theory of genetic algorithms. Early attempts [124, 125] were hardly successful, Rabani et al. [124] categorize GAs as "usually impossible to analyze". Among the reasons for this difficulty is a lack of deep understanding of the dynamics in populations as discussed earlier. Today's approaches are much less ambitious in that they do not aim at predicting the behavior of a general quadratical system but rather try to analyze concrete algorithms on specific problems.

Biological Background of Sexual Recombination. In biology it is as famous an open question as it is in evolutionary computation why sexual recombination should be advantageous. From the perspective of optimization, systems that adapt by mutation only are very efficient hill-climbers. In population genetics it is well-known that asexual reproduction permits efficient population increase and a fast adaption to the environment [6]. However, it is observed in some simple life forms that they may switch to sexual reproduction. This happens for example if their size, which decreases during asexual reproduction, falls below a threshold [74]. Among the negative effects of asexual reproduction is the reduction of biodiversity and gene variation. Sexual recombination on the other hand is considered more costly and could break up favorable gene combinations. But it can also have the opposite effect to break up negative correlations and it allows variation of the gene pool.

Schema Theory. In order to explain the success of crossover in evolutionary computation, early theoretical work on crossover was based on so-called Schema Theory, introduced by Holland [73]. A schema is a Boolean subspace where certain bits have fixed values and other bits can be set arbitrarily. Schema theory then deals with the growth of individuals belonging to schemas associated with high fitness values, in one generation of a GA. The building-block hypothesis states that using crossover GAs are able to combine good "building blocks", that is, schemata with few defining bits that contribute to a high fitness. However, there is neither a theoretical foundation nor convinc-

ing empirical evidence supporting this claim [167]: “An exact equation which generalizes the Holland Schema theorem can be proved, but like the Holland Schema theorem, it cannot be applied in realistic situations for more than one time step”, that is, it is very hard to make predictions for more than one generation of a GA.

Mixability. There are other approaches known within population genetics [61] and computer science [98] (published in the prestigious journal PNAS), developed independently of each other, to explain the advantages of sexual recombination. We discuss the work of Livnat et al. [98] who coined the term mixability. To match our notation we assume here that the genotype of an individual (which would correspond to a specific solution in combinatorial optimization) is given by a bitstring. A specific position within this bitstring with its concrete value is called an allele.
The authors use a standard model of population genetics with an infinite population which covers the whole search space. They discuss their model in comparison to asexual reproduction. In asexual reproduction already fit individuals will produce more offspring than less fit individuals, such that their relative frequency increases over time. For the population this has the effect that the average fitness increases. On the other hand in sexual recombination fit individuals are never able to pass on their genotype as a whole. In consequence this must mean that particularly fit individuals never survive with seemingly negative effects on the average fitness of the population. To explain this contradiction they define mixability with respect to an allele. For such an allele they compute an average fitness over all individuals which posses that specific allele. The authors argue that evolution is “survival of the most mixable” in contrast to the widely accepted hypothesis of the Darwinian “survival of the fittest” (maximization of the fitness).
There are several drawbacks to this approach. In combinatorial optimization we are never able to work with an infinite population. Additionally, the authors allow panmictic sexual reproduction, that is, any two parents can produce offspring. In population genetics this is a simplified assumption which ignores that mating usually only happens between carefully chosen parents. In combinatorial optimization this is reasonable only if we do not have any additional structural knowledge about the problem. Unfortunately, the experiments they perform are for very small genotypes only, which limits the significance of the results substantially.

After we introduce the state of the art in genetic algorithms in the following two paragraphs, we provide insights into the working principles of crossover as seen from the combinatorial optimization point of view. This defines the focus and research question of this thesis.

Evolutionary Computation. In contrast to the aforementioned concepts, practically used evolutionary algorithms employ a population (of finite size) that explore a fitness landscape defined by a given problem. In evolutionary computation crossover, also called recombination, is regarded an essential operator. Countless applications and empirical studies have shown that genetic algorithms with crossover are more effective than comparable algorithms using mutation only. While mutation typically makes only small changes to a solution, crossover can combine genetic material found in different solutions. The problem is that crossover is notoriously difficult to analyze. In order to handle the effect of crossover analytically, it is necessary to have a precise understanding of the dynamics within the population. However, the analysis of populations is much more difficult than the analysis of a (1+1) EA. Even on very simple problems like $\textsc{OneMax}(x) := \|x\|_1$ ($\|x\|_1$ denotes the number of ones), a problem well-understood and easily optimized by a (1+1) EA in $\Theta(n \log n)$ iterations, understanding the effect of crossover was a long standing open question, solved partially last year by Sudholt [146] for a restricted setting.

It is therefore not surprising that theoretical results on crossover after more than a decade of intensive research are still scarce. Speedups by crossover could be shown for coloring problems inspired by the Ising model [60, 145] and for the all-pairs shortest path problem [37, 44] as well as our results in Chapter 4. In terms of Pseudo-Boolean optimization, most results were actually limited to artificial functions [85, 131, 144, 160] constructed such that a theoretical analysis was possible.

A famous example of such an artificial Pseudo-Boolean function is the first proof that crossover leads to a speedup by Jansen and Wegener [86]. We state this function here, because we will revisit it in Chapter 3. The authors considered a simple test function $\textsc{Jump}_k$ and showed that uniform crossover can reduce a superpolynomial optimization time to a polynomial one. If $\|x\|_1$, again, denotes the number of ones in the bit string x, the function is defined as follows.

$$\textsc{Jump}_k(x) := \begin{cases} k + \|x\|_1 & \text{if } \|x\|_1 \leq (n-k) \text{ or } \|x\|_1 = n, \\ n - \|x\|_1 & \text{otherwise.} \end{cases}$$

The function leads the population of a genetic algorithm to a plateau of search points with equal fitness. All search points on the plateau have $n - k$ ones. The optimum can be generated by a uniform crossover in case there is a pair of sufficiently diverse parents in the population. This operation is far more efficient than using mutation to jump to the optimum; such a mutation has probability at most n^{-k} and takes n^k iterations on average.

Geometric Crossover. We define here the concept of geometric crossover which we use in Chapter 3. Let $d\colon M \times M \to \mathbb{R}$ be a metric for a set M . That is, for $x, y, z \in M$ the following properties hold: (i) $d(x, y) = 0 \Leftrightarrow x = y$, (ii) $d(x, y) = d(y, x)$, (iii) $d(x, y) + d(y, z) \geq d(x, z)$. Let the d-metric segment be given by the set $\{z \in M \mid d(x, z) + d(z, y) = d(x, y)\}$.

> **Definition 1.1** (Geometric crossover [107])**.** *A recombination operator (crossover) is a geometric crossover under the metric d if all offspring are in the d-metric segment between its parents.*

E. g. consider the *uniform crossover* where with probability $1/2$ it is decided which of the two parents passes on its specific bit to the offspring at position i. Under the Hamming H distance defined for two $x = x_1 \dots x_n, y = y_1 \dots y_n \in \{0, 1\}^n$ as $H(x, y) = \sum_{i=1}^{n} |x_i - y_i|$ the uniform crossover is a geometric crossover ([107]). In other words the uniform crossover chooses an offspring from the line segment between its parents. Hence, regarding the whole population uniform crossover will only produce offspring within the convex hull of the population.

Working Principles of Crossover. We conclude this section with some considerations on the working principles of crossover for an elitist environmental selection and give pointers to our own research within this thesis.

It is known that crossover reduces the optimization time on OneMax by a factor of two [146] albeit in the very special setting of a $(\mu+1)$ GA where only the fittest parents are allowed to recombine. Furthermore, the optimization time of genetic algorithms as compared to evolutionary algorithms drops from exponential to polynomial for the Real-Royal-Road functions [85]. There are some complexity theoretical reasons confirming the hypothesis that crossover speeds-up the optimization time in pseudo-Boolean optimization. In fact, in the theory of black-box optimization it is known that higher-arity operators such as crossover reduce the black-box optimization time on some classes of pseudo-Boolean functions [43].

Certainly, among the main advantages of crossover is its explorative effect within the search space. A standard bit mutation in pseudo-Boolean optimization which flips each bit with probability $1/n$, on average creates similar offspring with a constant Hamming distance. Geometric crossover like the uniform crossover on the other hand creates offspring in the d-segment such that the minimum Hamming distance to either parent x, y can be as large as $H(x, y)/2$. This explorative effect where crossover performs large jumps in the

search space and mutation explores the closer neighborhood is discussed and proven in Chapter 4, cf. Section 4.3. In order to perform well, crossover depends on diversity in the population. In Chapter 4 this is naturally given due to the representation of the combinatorial optimization problem. However, in the context of pseudo-Boolean optimization in Chapter 3 we see that high crossover rates have negative effects, simultaneously, that need to be carefully balanced by the mutation rate as well as the population size.

Technical Machinery. There have been some impressive developments of the mathematical tools available for the analysis in evolutionary computation [120]. Apart from the early use of Markov Chains, analyses include classical tools from probability theory such as the coupon collector's problem, gambler's ruin, as well as probabilistic tail bounds and inequalities like Markov's inequality and Chernoff bounds [58, 108]. There are also a number of techniques that have been developed by the evolutionary computation community in order to fit their needs. By now classical examples are the analysis of fitness levels and typical runs. Newer developments are family trees and drift-analysis, see [120] for a detailed overview. Since the publication of [120] research on generic analysis methods has concentrated on drift analysis [29, 41, 45, 96, 118, 119], which is a special case of the theory on martingales. Drift analysis has led to more concise analyses of EAs but has also given rise to some results on populations [20, 95]. With these developments a recent trend has started to use martingales to handle more involved problems.

Some Final Remarks. We remind the reader of the fact that unless stated otherwise we always measure the runtime of a stochastic search algorithm by the number of iterations until the termination condition is met. This is called *optimization time*. We usually disregard additional overhead and assume that a call to the black-box, that is an evaluation of the fitness function, is the most expensive step within an iteration.

By *iff* we denote the mathematical term if and only if, and by u. a. r., uniformly at random, that is some event happens with uniform distribution.

Finally, the term w. h. p. (with high probability) denotes a result that holds with probability at least $(1 - O(n^{-c}))$ for some constant $c > 0$ independent of n.

1.3 Outline

This thesis – apart from the introduction – is divided into three chapters which are largely self-contained.

> The contribution of each chapter is discussed after the necessary notations are introduced. They can be found in Sections 2.1, 3.1, and 4.3.

In Chapter 2 we consider the large class of problems that have a dynamic programming formulation. We develop evolutionary algorithms for this class and prove optimization bounds. While the main focus of this thesis is in the investigation of crossover operators this chapter is a success in its own right. This is the second time that for a large class of combinatorial optimization problems theoretical insights were gained. In Section 2.5 we discuss some experiments using crossover for the Traveling Salesperson problem based on its dynamic programming representation. Chapter 3 discusses some working principles of crossover in the setting of pseudo-Boolean optimization by analyzing the two functions OneMax and $\textsc{Jump}_k$. Our theoretical findings are supplemented by Monte-Carlo simulations in Section 3.4 in the regime not covered by our theorems. In Chapter 4 we discuss several different aspects of crossover by means of analyzing a $(\mu+1)$ GA for the combinatorial optimization problem All-Pairs Shortest Path. In Section 4.4 we state an upper and lower bound for the algorithm. In Section 4.5 we discuss repair strategies for our crossover operators. Our theorems are proved in Section 4.6. A further extension in terms of a multi-criteria scenario is given in Section 4.7. And finally, we conclude with a discussion about the insights gained in Chapter 5.

2

Evolutionary Algorithms for the Class of DP-Problems

Among the challenges in the theory of evolutionary computation is the understanding under which conditions combinatorial optimization problems can be tackled by evolutionary algorithms. It is straightforward to use these type of algorithms as easy-to-implement heuristics. But our focus goes beyond their mere use, we would like to analyze how these problems can be solved with generic approaches. It is thus important to understand how their components like particular representations or variation operators work. Finally, we would like to develop methods to analyze evolutionary algorithms, to understand when and why these algorithms are able to compute (optimal) solutions to a problem. Our aim is to give theoretical guarantees for measures well-known from algorithm theory like runtime bounds or approximation ratios.

At the beginning of the theory of evolutionary algorithms it has been common to design *artificial* test functions to make them accessible to theoretical analysis and to demonstrate a desired property. Among the initial successes were theoretical analyses for some of those function-classes. For example Droste et al. [49] and Wegener and Witt [162] prove upper and lower bounds for prototypical algorithms like the (1+1) EA on the class of linear, unimodal and monotone functions. Of course, the design of an appropriate evolutionary algorithm and its analysis get harder the more realistic a problem becomes. This is especially true when real-world combinatorial optimization problems are involved. Despite the technical difficulties there have been a number of analyses accomplished during the last decade. However, a recurring theme is that none of these analyses covers more than a specific problem. Techniques developed in this process only rarely carry over to other problems.

A notable exception and to the best of our knowledge the first of its kind is the work of Reichel and Skutella [129], where the authors analyze the performance of evolutionary algorithms on a very general class of problems, namely matroid optimization problems. Thus, unifying some known solutions for problems like maximum bipartite matching and minimum spanning trees. The authors provide runtime guarantees, and depending on the type of matroid optimization they prove that a problem is either solved to optimality or approximated within a given ratio.

A huge benefit of the approach taken in [129] is the natural representation of the ground set (of the matroid) as bitstrings, a clever design of a penalty fitness function and the solution with a prototypical (1+1) EA or Randomized Local Search (RLS) employing the commonly used mutation operator on bitstrings.

This discussion above points out the decisions that have to be taken when designing evolutionary algorithms for the solution of combinatorial optimization problems. We need to choose a representation for a problem solution, the variation as well as selection operators for a specific type of evolutionary algorithm, e. g. (1+1) EA or RLS.

In this part of the thesis we contribute to the theoretical understanding of evolutionary algorithms for another large class of combinatorial optimization problems. This is the second time in the theory of evolutionary algorithms that we are able to unify a number of different analyses for a number of singular problems into a larger theory. We design evolutionary algorithms for the class of problems that permit a dynamic programming approach and give algorithmic guarantees for the solution of these problems.

2.1 Contribution

Dynamic programming (DP) [8] is a well-known algorithmic technique that helps to tackle a wide range of problems. A general framework for dynamic programming has been considered by e. g. Woeginger [165] and Klötzler [88]. The technique allows to compute an optimal solution for the problem by extending partial solutions to an optimal one.

Recently, it has been proven for various combinatorial optimization problems with a dynamic programming solution that they can be solved by evolutionary algorithms in reasonable time using a suitable representation together with mutation operators adjusted to the given problem. Examples for this approach are the single source shortest path problem [138], all-pairs short-

est path problem [32, 44], multi-objective shortest path problem [75, 111], the traveling salesman problem [150] and the knapsack problem [55]. The representations used in these papers are different from the general encodings working with binary strings as considered earlier in theoretical works on the runtime behavior of evolutionary algorithms. Instead, the chosen representations reflect some properties of partial solutions of the problem at hand that allow to obtain solutions that can be extended to optimal ones for the considered problem. To obtain such partial solutions the algorithms make use of diversity mechanisms allowing the algorithms to proceed in a dynamic programming way.

An important common feature of the aforementioned evolutionary algorithms is that each of them is based on a suitable multi-objective formulation of the given problem. The schemes of these EAs and solution representations are different, however.

We unify all of these approaches into a single theory. We describe how to *represent possible solutions* such that the search process becomes provably efficient. This is one of the main questions when designing an evolutionary algorithm for a given problem.

How to find the best representation for a given problem has been extensively studied in the literature on evolutionary algorithms [133]; for example there are different representations for the well-known traveling salesman problem (see e. g. Michalewicz [102]) or the NP-hard spanning tree problem (see e. g. Raidl and Julstrom [127]). Each of these representations induces a different neighborhood of a particular solution, and variation operators such as mutation and crossover have to be adjusted to the considered representation. Usually, such representations either lead directly to feasible solutions for the problem to be optimized or the search process is guided towards valid solutions by using some penalty functions. Thus, the representation of possible solutions in combination with some suitable variation operators is often crucial for the success of the algorithm.

Some of the problems considered in our literature discussion are tackled by evolutionary algorithms that use a representation which enables them to construct solutions in a dynamic programming fashion. We take a general approach and relate the construction of such algorithms to the development of algorithms using dynamic programming techniques. Thereby, we give general guidelines on how to develop evolutionary algorithms that have the additional ability of carrying out dynamic programming steps.

Each gene in our problem representation defines one of the DP transition mappings, and a composition of these mappings yields the DP state represented

by the individual. The proposed EA utilizes a mutation operator which is a special case of point mutation, where the gene subject to change is not chosen uniformly at random as usual, but selected as the first gene which has never been mutated so far (see Section 2.4.2 for details and links to the biological systems).

The goal of this chapter is to relate the above mentioned multi-objective evolutionary approaches to dynamic programming and give a general setup for evolutionary algorithms that are provably able to solve problems having a dynamic programming formulation. In particular, we show that in many cases a problem that can be solved by dynamic programming in time T has an evolutionary algorithm which solves it in expected runtime $O(T \cdot n \cdot \log(|DP|))$ with n being the number of phases and $|DP|$ being the number of states produced at the completion of dynamic programming. Here, we discuss the expected runtime as opposed to the usually used measure expected optimization time that only counts fitness evaluations and ignores the overhead computation.

The obtained results are not aimed at the development of faster solution methods for these combinatorial optimization problems (to construct an EA in our framework, one has to know enough about the problem so that the traditional DP algorithm could be applied and this algorithm would be more efficient). Instead, we aim at characterizing the area where evolutionary algorithms can work efficiently and study the conditions that ensure this. To put it informally, our results imply that a class of problems that is easy for the DP algorithm is also easy for a suitable EA for most of the reasonable meanings of the term "easy" (solvable in polynomial or pseudo-polynomial running time or admitting an FPTAS (see Definition 4.32)).

The insight that a general framework for problems solvable by dynamic programming was possible due to the authors own work on the Traveling Salesperson Problem in [150]. At very much the same time this thought appeared to Benjamin Doerr, Anton Eremeev, Frank Neumann, and Christian Thyssen. Hence, we joined forces to publish [38, 40]. This chapter is based on these publications – except for the experiments presented in Section 2.5 that haven't been published elsewhere.

Remark. We remark that a large class of so-called DP-benevolent problems admits a fully polynomial time approximation scheme (FPTAS) following the ideas presented by Woeginger [165]. For this class, the authors of [40] on which this chapter is based, develop a fully polynomial time randomized approximation scheme (FPRAS). These results are mainly due to one coauthor [56]. Hence, they are omitted here.

Organization

The rest of the chapter is organized as follows. In Section 2.2, we introduce a general dynamic programming formulation and the kind of problems that we want to tackle. This dynamic programming approach is transferred into an evolutionary algorithm framework in Section 2.4. In Section 2.3 and 2.4.5 we also show how to obtain evolutionary algorithms carrying out dynamic programming for some well-known combinatorial optimization problems.

The main results of this chapter originally were sketched in the extended abstract [38]. Additionally to the refined presentation of results [38], this chapter contains a DP-based EA with a strengthened runtime bound for the case of DP algorithm with homogeneous transition functions (applicable e.g. to the shortest path problems) which were published in the journal version [40].

2.2 Dynamic Programming

Dynamic programming is a general design paradigm for algorithms. The basic idea is to divide a problem into subproblems of the same type, and to construct a solution for the whole problem using the solutions for the subproblems. Dynamic programming has been proven to be effective for many single-objective as well as multi-objective optimization problems. It is even the most efficient approach known for some problems in scheduling [123, 165], bioinformatics [22], routing (see e.g. [23, Chapters 24 and 25]) and other areas.

In this section, we will assume that an original optimization problem Π (single-objective or multi-objective) may be transformed into a multi-objective optimization problem P of a special type. It is easy to see that this transformation is indeed reasonable as in dynamic programming we need some means to insert the optimization goal or some constraints into the dynamic program. This increases the dimension of the problem, for example consider the famous dynamic program for knapsack discussed in Section 2.3 along with several other examples of the transformation from Π to P. Thus, the general scheme of dynamic programming will be presented and studied here in terms of the problem P.

2.2.1 Multi-Objective Optimization Problem

This section introduces the notation for multi-objective dynamic programming problems. We remark that here we define a generic approach to use an evolutionary algorithm for these problems. The representation and solution for specific problems can of course differ. To this end we refer the reader to Section 4.7.1 where we treat the problem-specific representation and solution of the multi-criteria All-Pairs Shortest Path problem (APSP) that also has a dynamic programming structure. However, all results stated in this chapter also hold if we would tackle the multi-criteria APSP with an evolutionary algorithm.

Let us consider a multi-objective optimization problem P which will be well suited for application of the DP algorithm in some sense, as shown below. We will discuss single-objective optimization as a special case in Section 2.2.3.

Assume there are $d \in \mathbb{N}$ objectives that have to be optimized in P. An instance of problem P is defined by a quadruple $(d, g, \mathcal{S}, \mathcal{D})$. Here $\mathcal{S}$ is called *search space*, and in what follows, it is supposed that $\mathcal{S} \subseteq \mathcal{S}'$, where $\mathcal{S}'$ is a superset of $\mathcal{S}$, which also contains all infeasible states that may arise. The mapping $g\colon \mathcal{S}' \to (\mathbb{R} \cup \{\infty\})^d$ is called the *objective function*, and $g(\mathcal{S}') \subseteq (\mathbb{R} \cup \{\infty\})^d$ is the *objective space*, in which infeasible states are mapped to $(\infty)^d$. $\mathcal{D} \subseteq \mathcal{S}$ is a set of feasible solutions.

We introduce the following partial order to define the goal in multi-objective optimization formally. Throughout this thesis, $\preceq$ denotes *Pareto dominance* where

$$(y_1, \ldots, y_d) \preceq (y'_1, \ldots, y'_d)$$

iff $y_i \geq y'_i$ for all i for minimization criteria g_i and $y_i \leq y'_i$ for maximization criteria g_i. In the following, we use the notation $y' \prec y$ as an abbreviation for $y' \preceq y$ and $y \npreceq y'$. The *Pareto front* is the subset of $g(\mathcal{D})$ that consists of all maximal elements of $g(\mathcal{D})$ with respect to $\preceq$. The goal is to determine the *Pareto-optimal set*, that is, an inclusion-minimal subset of feasible solutions $\mathcal{D}$ that is mapped on the Pareto front by g.

We remark that the Pareto optimal set might be very large such that from the practical point of view an approximation seems to be more suitable. However, in the theory of dynamic programming we are first of all interested to show that a problem can be solved to optimality.

2.2.2 Framework for Dynamic Programs

Consider a DP algorithm for a problem P, working through a number of iterations called *phases*. In each phase the DP algorithm constructs and stores some *states* belonging to $\mathcal{S}$. By saying that DP algorithm computes the Pareto-optimal set for the problem P we mean that after completion of the DP algorithm, the set of all DP states produced at the final phase is the Pareto-optimal set for P.

Application of the DP approach to many multi-objective and single-objective optimization problems can be viewed as a transformation of a given problem Π to some problem P: a DP algorithm is applied to compute the Pareto-optimal set for P and this set is efficiently transformed into a solution to the given single- or multi-objective problem.

In what follows, we consider only those DP algorithms where the states of the current phase are computed by means of *transition functions*, each such function depending on the input parameters of problem P and taking as an argument some state produced at the previous phase.

Let us start the formal definition of the DP algorithm from a simplified version. Suppose that the simplified DP algorithm works in n phases, such that in the i-th phase a set $\mathcal{S}_i \subseteq \mathcal{S}$ of states is created. We use n finite sets $\mathcal{F}_i$ of state transition functions $F\colon \mathcal{S} \to \mathcal{S}'$ to describe the DP algorithm. A mapping F can produce elements $F(S) \in \mathcal{S}' \backslash \mathcal{S}$ that do not belong to a search space. To discard such elements at phase i, $i = 1, \ldots, n$, a *consistency function* H_i is used, $H_i\colon \mathcal{S}' \to \mathbb{R}$, such that $S \in \mathcal{S}$ if and only if $H_i(S) \leq 0$. We assume that the number n, the functions H_i and the sets of functions $\mathcal{F}_i$ depend on the input instance of problem P.

The simplified DP algorithm proceeds as follows. In the initialization phase, the state space $\mathcal{S}_0$ is initialized with a finite subset of $\mathcal{S}$. In the i-th phase, the state space $\mathcal{S}_i$ is computed using the state space $\mathcal{S}_{i-1}$ according to

$$\mathcal{S}_i = \{F(S) \mid S \in \mathcal{S}_{i-1} \wedge F \in \mathcal{F}_i \wedge H_i(F(S)) \leq 0\}. \tag{2.1}$$

In the process, the consistency functions H_i serve to keep the infeasible elements emerging in phase i from being included into the current state space $\mathcal{S}_i$. (Note that after completion of phase n of the simplified DP algorithm, the set $\mathcal{S}_n$ may contain some states whose objective values are Pareto-dominated by those of other states from $\mathcal{S}_n$.)

To delete the states with Pareto-dominated objective values and to improve the runtime of the simplified DP algorithm defined by (2.1), most of the

practical DP algorithms utilize the Bellman principle (see e. g. [8]) or its variations so as to dismiss unpromising states without affecting the optimality of the final set of solutions. A formulation of the Bellman principle in terms of recurrence (2.1) for the single-objective problems can be found in Section 2.2.3. Sufficient conditions for application of the Bellman principle in the single-objective case were formulated in [106]. In the multi-objective case the Bellman principle is not used, but the unpromising states may be excluded by means of an appropriate dominance relation on the set of states. Originally such dominance relations were introduced by Klötzler [88]. Here, we employ a similar approach, motivated by [165].

Let us consider a partial quasi-order (i. e. a reflexive and transitive relation) $\preceq_{\text{dom}}$ defined on $\mathcal{S}$ so that $S \preceq_{\text{dom}} S'$ iff $g(S) \preceq g(S')$. We will say that state S is *dominated* by state S' iff $S \preceq_{\text{dom}} S'$. If $S \in \mathcal{T} \subseteq \mathcal{S}$ is such that no $S' \in \mathcal{T}$ exists satisfying $S \preceq_{\text{dom}} S'$, then S will be called *non-dominated* in $\mathcal{T}$.

As we will see, under the following two conditions the relation $\preceq_{\text{dom}}$ is helpful to dismiss unpromising states in the DP algorithm.

The first Condition 2.1 guarantees that the dominance relation between two states transfers from one round to the next:

> **Condition 2.1.** *For any* $S, S' \in \mathcal{S}_{i-1}, i = 1, \ldots, n$, *if* $S \preceq_{\text{dom}} S'$ *then* $F(S) \preceq_{\text{dom}} F(S')$ *for all* $F \in \mathcal{F}_i$.

The second Condition 2.2 expresses that infeasible states cannot dominate feasible states:

> **Condition 2.2.** *For any* $S, S' \in \mathcal{S}'$, *if* $S \preceq_{\text{dom}} S'$ *and* $H_i(S) \leq 0$ *then* $H_i(S') \leq 0$.

Consider a subset $\mathcal{S}_i$ of $\mathcal{S}$. We call $\mathcal{T}_i \subseteq \mathcal{S}_i$ a *dominating subset* of $\mathcal{S}_i$ with respect to $\preceq_{\text{dom}}$ iff for any state $S \in \mathcal{S}_i$ there is a state $S' \in \mathcal{T}_i$ with $S \preceq_{\text{dom}} S'$. Let us use the notation $M(\mathcal{S}_i, \preceq_{\text{dom}})$ to denote the set of all dominating subsets of $\mathcal{S}_i$ which are minimal by inclusion.

The following proposition indicates that under Conditions 2.1 and 2.2 it is sufficient to keep a dominating subset of states constructed in each phase i, rather than the full subset $\mathcal{S}_i$.

Proposition 2.3. *Suppose the simplified DP algorithm is defined by (2.1), Conditions 2.1 and 2.2 hold and the dominating sets* $\mathcal{T}_i,\ i = 1, \ldots, n$ *are computed so that* $\mathcal{T}_0 \in M(\mathcal{S}_0, \preceq_{\mathrm{dom}})$,

$$\mathcal{T}_i \in M(\{F(S) \mid S \in \mathcal{T}_{i-1} \wedge F \in \mathcal{F}_i \wedge H_i(F(S)) \leq 0\}, \preceq_{\mathrm{dom}}). \tag{2.2}$$

Then for any state $S^* \in \mathcal{S}_i,\ i = 0, \ldots, n$, *there exists* $S \in \mathcal{T}_i$ *such that* $S^* \preceq_{\mathrm{dom}} S$.

Proof. The proof is by induction on i. For $i = 0$ the statement holds by assumption $\mathcal{T}_0 \in M(\mathcal{S}_0, \preceq_{\mathrm{dom}})$.

By Equation (2.1), a state $S^* \in \mathcal{S}_i$ can be expressed as $S^* = F^*(S')$, so that $H_i(S^*) \leq 0$, $F^* \in \mathcal{F}_i$ and $S' \in \mathcal{S}_{i-1}$. But by induction hypothesis, there exists a state $S^\diamond \in \mathcal{T}_{i-1}$ such that $S' \preceq_{\mathrm{dom}} S^\diamond$. Now Conditions 2.1 and 2.2 imply that $S^* = F^*(S') \preceq_{\mathrm{dom}} F^*(S^\diamond)$ and $F^*(S^\diamond) \in \{F(S) \mid S \in \mathcal{T}_{i-1} \wedge F \in \mathcal{F}_i \wedge H_i(F(S)) \leq 0\}$. Hence, by (2.2) we conclude that there exists $S \in \mathcal{T}_i$ such that $S^* \preceq_{\mathrm{dom}} F^*(S^\diamond) \preceq_{\mathrm{dom}} S$. □

In view of definition of $\preceq_{\mathrm{dom}}$, if the conditions of Proposition 2.3 are satisfied and the Pareto front of g is contained in $g(\mathcal{S}_n)$, then this Pareto front is also contained in $g(\mathcal{T}_n)$.

Proposition 2.4. *If the conditions of Proposition 2.3 are satisfied, then the size of each* $\mathcal{T}_i,\ i = 0, \ldots, n$, *is uniquely determined.*

Proof. Consider the set of maximal elements of $\mathcal{S}_i$ with respect to $\preceq_{\mathrm{dom}}$. Define the equivalence classes of this set with respect to the equivalence relation $x \equiv y$ iff $x \preceq_{\mathrm{dom}} y$ and $y \preceq_{\mathrm{dom}} x$. The size of a minimal subset M of the set of maximal elements, which dominates all elements of $\mathcal{S}_i$, is unique since such M contains one representative element from each equivalence class. □

A computation satisfying (2.2) can be expressed in an algorithmic form as presented in Algorithm 2.1. It is easy to see that when a subset $\mathcal{T}_i$ is completed in Lines 6–10, condition (2.2) holds.

The runtime of a DP algorithm depends on the computation times for the state transition functions $F \in \mathcal{F}_i$, for the consistency functions H_i, for checking the dominance and manipulations with the sets of states. Let θ_F be an upper

Algorithm 2.1: Dynamic Program for P

```
1  T_0 ← ∅;
2  for S ∈ S_0 do
3      if ∄S' ∈ T_0 : S ⪯_dom S' then
4          T_0 ← (T_0 \ {S' ∈ T_0 | S' ≺_dom S}) ∪ {S};
5  for i = 1 to n do
6      T_i ← ∅;
7      foreach S ∈ T_{i-1} do
8          foreach F ∈ F_i do
9              if H_i(F(S)) ≤ 0 and ∄S' ∈ T_i : F(S) ⪯_dom S'
               then
10                 T_i ← (T_i \ {S' ∈ T_i | S' ≺_dom
                   F(S)}) ∪ {F(S)};
11 return T_n;
```

bound on computation time for a transition function F and let $\theta_{\mathcal{H}}$ be an upper bound for computation time of any function $H_i,\ i = 1, \ldots, n$. Sometimes it will be appropriate to use the average computation time for the state transition functions at phase $i,\ i = 1, \ldots, n$: $\theta_{\mathcal{F}_i} = \sum_{F \in \mathcal{F}_i} \theta_F / |\mathcal{F}_i|$.

In Algorithm 2.1, verification of condition

$$\nexists S' \in \mathcal{T}_i \colon F(S) \preceq_{\text{dom}} S' \tag{2.3}$$

in Line 9 and execution of Line 10 may be implemented using similar problem-specific data structures. To take this into account, we will denote by $\theta_{\preceq}$ an upper bound applicable both for the time to verify (2.3) and for the time to execute Line 10.

The body (Lines 9–10) of the main loop (Lines 5–10) in Algorithm 2.1 is executed $\sum_{i=1}^{n} |\mathcal{T}_{i-1}| \cdot |\mathcal{F}_i|$ times.

To simplify the subsequent analysis let us assume that in the if-statement at Line 9, the condition (2.3) is always checked. We denote the computation time for initializing $\mathcal{T}_0$ with θ_{ini} (Lines 1–4) and the computation time for presenting the result with θ_{out} (Line 11), which leads to an overall runtime

$$O\left(\theta_{\text{ini}} + \sum_{i=1}^{n} |\mathcal{F}_i| \cdot |\mathcal{T}_{i-1}| \cdot (\theta_{\mathcal{F}_i} + \theta_{\mathcal{H}} + \theta_{\preceq}) + \theta_{\text{out}}\right). \tag{2.4}$$

In many applications of the DP, the computation time for the state transition functions and the consistency functions are constant. Besides that, the partial

quasi-order $\succeq_{\text{dom}}$ is often just a product of linear orders and it is sufficient to allocate one element in a memory array to store one (best found) element for each of the linear orders. This data structure usually allows to verify (2.3) and to execute Line 10 in constant time (see the examples in Subsection 2.4.5). In the cases mentioned above, the values θ_F, $\theta_{\mathcal{H}}$ and $\theta_{\preceq}$ can be chosen equal to the corresponding computation times and the overall DP algorithm runtime in (2.4) can be expressed with symbol $\Theta(\cdot)$ instead of $O(\cdot)$.

Note that the runtime of the DP algorithm is polynomially bounded in the input length of problem P if θ_{ini}, n, $\theta_{\mathcal{F}_i}$, $\theta_{\mathcal{H}}$, $\theta_{\preceq}$, θ_{out}, as well as $|\mathcal{T}_i|$ and $|\mathcal{F}_{i+1}|$ for $i = 0, \ldots, n-1$, are polynomially bounded in the input length. Here and below, we say that a value (e.g. the running time) is polynomially bounded in the input length, meaning that there exists a polynomial function of the input length, which bounds the value from above.

2.2.3 Bellman Principle for Single-Objective Problems

Here, we describe the Bellman optimality principle in terms of the DP method defined by recurrence (2.1). Consider a single-objective maximization problem Π. Let $2 \leq \beta \in \mathbb{N}$ be a constant and $\mathcal{S} \subseteq \mathbb{R}^\beta$, so that the first component s_1 of a state $S \in \mathcal{S}$ characterizes a quality of the state in some sense.

The Bellman principle applies to a DP algorithm for Π if the following statement holds. Suppose that starting from some state $S_0^* \in \mathcal{S}_0$, a sequence of "decisions" $F_1 \in \mathcal{F}_1, \ldots, F_n \in \mathcal{F}_n$ leads to an optimal solution for Π. Let us denote $S_i^* = F_i(F_{i-1}(...F_1(S_0^*)...)) \in \mathcal{S}_i,\ i = 1, \ldots, n$. Then for any particular state $S_i^* = (s_{1i}^*, \ldots, s_{\beta i}^*)$, the subsequence $F_1, \ldots, F_i$ is an optimal policy for reaching the set of states coinciding with S_i^* in components $s_2, \ldots, s_\beta$. By an optimal policy here we mean that for any sequence $F_1' \in \mathcal{F}_1, \ldots, F_i' \in \mathcal{F}_i$ starting with some $S_0' \in \mathcal{S}_0$, such that $S_k' = F_k'(F_{k-1}'(...F_1'(S_0')...)) \in \mathcal{S}_k,\ k = 1, \ldots, i$, and $S_i' = (s_{1i}', s_{2i}^*, \ldots, s_{\beta i}^*)$, holds $s_{1i}' \leq s_{1i}^*$.

If the Bellman principle applies to a DP algorithm, then for any $s_2, \ldots, s_\beta$ it is possible to keep only one state which dominates all states in the subset $\{S' \in \mathcal{S}_i \ :\ s_2' = s_2, \ldots, s_\beta' = s_\beta\}$ without a risk to loose optimality of the DP algorithm result.

2.3 Applications of the General DP Scheme

In this subsection, we point out how the general DP framework presented above is applied to some classical combinatorial optimization problems. The approach followed here is to describe the appropriate problem P and the components of a dynamic programming algorithm for the solution of a specific problem Π. Most of the following examples have been inspired by the previous works [44, 138] as well as the authors own work [150] discussed in the subsequent paragraph. Note that $\preceq_{\text{dom}}$ will be a product of linear orders in each of these examples. In what follows id denotes the identical mapping.

Traveling Salesman Problem. Let us first consider the traveling salesman problem (TSP) as a prominent NP-hard example. The input for TSP consists of a complete graph $\mathcal{G} = (V, E)$ with a set of nodes $V = \{1, 2, \ldots, n\}$ and non-negative edge weights $w\colon E \to \mathbb{R}_0^+$. It is required to find a permutation of all nodes $(v_1, \ldots, v_n)$, such that the TSP tour length $\sum_{i=2}^{n} w(v_{i-1}, v_i) + w(v_n, v_1)$ is minimized. Without loss of generality we can assume that $v_1 = 1$, that is, the TSP tour starts in the fixed vertex 1.

The search space $\mathcal{S}$ for problem P corresponding to the dynamic programming algorithm of Held and Karp [72] consists of all paths $S = (v_1, \ldots, v_i), v_1 = 1$ of $i = 1, \ldots, n$ nodes. $\mathcal{S}'$ is a superset of the search space which consists of all sequences of nodes up to length n (the same node may occur more than once) which are infeasible. Given $M \subseteq V\backslash\{1\}$ and $k \in M$, let $\pi(k, M)$ denote the set of all paths of $|M|+1$ vertices starting in vertex 1 then running over all nodes from M and ending in vertex k. Let the vector objective function $g\colon \mathcal{S}' \to (\mathbb{R} \cup \{\infty\})^d$, $d = (n-1)2^{(n-1)}$ have components $g_{kM}(S)$ for all $M \subseteq V\backslash\{1\}$, $k \in M$, equal to the length of path S iff $S \in \pi(k, M)$. For all other $S \notin \pi(k, M)$ assume $g_{kM}(S) = \infty$. The set of feasible solutions is $\mathcal{D} = \cup_{k=2}^{n} \pi(k, V\backslash\{1\})$, since in the TSP we seek a tour of length n.

$\mathcal{S}_0$ consists of a single element v_1. The set $\mathcal{F}_i$ for all i consists of $n-1$ functions $F_v\colon \mathcal{S} \to \mathcal{S}'$ that add vertex $v \in V\backslash\{1\}$ to the end of the given path. For invalid states $S \in \mathcal{S}'$, which are characterized by not being Hamiltonian paths on their vertex sets, the mapping $H_i(S)$ computes 1 and 0 otherwise.

In view of the definition of objective g, the dominance relation for $S, S' \in \mathcal{S}$ is formulated as follows. $S \preceq_{\text{dom}} S'$ if and only if S and S' are Hamiltonian paths on the same ground set with the same end vertex k and path S' is not longer than S. States from different sets $\pi(k, M)$ are not comparable. Conditions 2.1 and 2.2 are verified straightforwardly.

Substituting these components into Algorithm 2.1, we almost get the well-known dynamic programming algorithm of Held and Karp [72], except for the last step where the optimal tour is constructed from the optimal Hamiltonian paths.

Algorithm 2.1 initializes the states of the dynamic program with paths $(1, v)$ for all $v \in V \setminus \{1\}$. In each subsequent iteration i, the algorithm takes each partial solution S obtained in the preceding iteration and checks for every application of the state transition function $F(S)$ with $F \in \mathcal{F}_i$ whether $H_i(F(S))$ is a feasible partial solution that is non-dominated in $\mathcal{T}_i$. If so, then $F(S)$ is added to the set $\mathcal{T}_i$ of new partial solutions by replacing dominated partial solutions S' defined on the same ground set with the same end vertex of the Hamiltonian path.

What remains to do after completion of the DP algorithm with Pareto-optimal set is to output the Pareto-optimal solution minimizing the criterion $g_{k,V\setminus\{1\}}(S) + w(k,1), k \in V\setminus\{1\}$, which is now easy to find. Here using appropriate data structures one gets $\theta_F = \Theta(1)$, $\theta_{\mathcal{H}} = \Theta(1)$, $\theta_{\preceq} = \Theta(1)$ and $|\mathcal{T}_i| = i\binom{n-1}{i}$, $|\mathcal{F}_i| = n-1$ for all $i = 1, \ldots, n$, thus the observation following (2.4) leads to the time complexity bound $O\big(n^2 2^n\big)$.

Knapsack Problem. Another well-known NP-hard combinatorial optimization problem that can be solved by dynamic programming is the knapsack problem. The input for the knapsack problem consists of n items where each item i has an associated integer weight $w_i > 0$ and profit $p_i > 0$, $1 \le i \le n$. Additionally a weight bound W is given. The goal is to determine an item selection $K \subseteq \{1, \ldots, n\}$ that maximizes the profit $\sum_{i\in K} p_i$, subject to the condition $\sum_{i\in K} w_i \le W$.

We fit the problem into the above framework assuming that each state $S = (s_1, s_2) \in \mathcal{S}_i,\ i = 1, \ldots, n$, encodes a partial solution for the first i items, where coordinate s_1 stands for the weight of a partial solution and s_2 is its profit. The initial set $\mathcal{S}_0$ consists of a single element $(0, 0)$ encoding a selection of no items.

The pseudo-Boolean vector function $g\colon \mathcal{S}' \to \mathbb{R}^W$ defines W criteria

$$g_w(S) := \begin{cases} s_2 & \text{if } s_1 = w \\ 0 & \text{otherwise} \end{cases}, \quad w = 0, \ldots, W, \tag{2.5}$$

that have to be maximized. This implies the dominance relation $\preceq_{dom}$ such that $S \preceq_{\text{dom}} S'$ iff $s_1 = s_1'$ and $s_2 \le s_2'$, where $S = (s_1, s_2),\ S' = (s_1', s_2')$.

The set $\mathcal{F}_i$ consists of two functions: id and $F_i(s_1, s_2) = (s_1 + w_i, s_2 + p_i)$. Here F_i corresponds to adding the i-th item to the partial solution, and id corresponds to skipping this item. A new state $S = (s_1, s_2)$ is accepted if it does not violate the weight limit, i. e. $H_i(S) \leq 0$, where $H_i(S) = s_1 - W$.

The Conditions 2.1 and 2.2 are straightforwardly verified. To obtain an optimal solution for the knapsack problem it suffices to select the Pareto-optimal state with a maximal component s_2 from $\mathcal{S}_n$.

To reduce the comparison time $\theta_{\preceq}$ we can store the states of the DP in a $(W \times n)$-matrix. An element in row w, $w = 1, \ldots, W$, and column i, $i = 1, \ldots, n$, holds the best value s_2 obtained so far on states $S = (s_1, s_2) \in \mathcal{T}_i$ with $s_1 = w$. Then $\theta_{\preceq}$ is a constant and the worst-case runtime of the explained DP algorithm is $O(n \cdot W)$ since $\sum_{i=1}^{n} |\mathcal{T}_{i-1}| \leq nW$.

Single Source Shortest Path Problem. A classical problem that also fits into the DP framework is the single source shortest path problem (SSSP). Given an undirected connected graph $\mathcal{G} = (V, E)$, $|V| = n$ and positive edge weights $w \colon E \to \mathbb{R}^+$, the task is to find shortest paths from a selected *source* vertex $s \in V$ to all other vertices.

The search space $\mathcal{S}$ is a set of all paths in $\mathcal{G}$ with an end-point s. The set of feasible solutions $\mathcal{D}$ is equal to $\mathcal{S}$.

Since adding a vertex to a path may result in a sequence of vertices that do not constitute a path in $\mathcal{G}$, we define the set $\mathcal{S}' \supseteq \mathcal{S}$ to consist of all sequences of vertices of length at most n with an end-point s. The set $\mathcal{S}_0$ of initial solutions is just a single vertex s. Now for all i, we define $\mathcal{F}_i := \{F_v \mid v \in V\} \cup \{\mathrm{id}\}$, where $F_v \colon \mathcal{S} \to \mathcal{S}'$ is the mapping adding the vertex v to a sequence of vertices. $H_i(S) = -1$ if S is a path in $\mathcal{G}$ with an end-point s, and 1 if not.

Let the vector objective function g have $d = n$ components $g_v(S)$ for all $v \in V$, equal to the length of path S iff S connects s to v, otherwise assume $g_v(S) = \infty$. This implies that for two paths $S, S' \in \mathcal{S}$ we have $S \preceq_{\mathrm{dom}} S'$ if and only if the paths S and S' connect s to the same vertex and S' is not longer than S.

The resulting DP algorithm has $\theta_F = \Theta(1)$, $\theta_{\mathcal{H}} = \Theta(1)$, $\theta_{\preceq} = \Theta(1)$ and $|\mathcal{T}_{i-1}| = \Theta(n)$, $|\mathcal{F}_i| = \Theta(n)$ for all $i = 1, \ldots, n$, thus (2.4) gives the time complexity bound $O(n^3)$. The well-known Dijkstra's algorithm has $O(n^2)$ time bound, but in that algorithm only one transition mapping is applied in each phase (attaching the closest vertex to the set of already reached ones), and such a problem-specific DP scheme is not considered here.

All-Pairs Shortest Path Problem. Finally, let us consider the all-pairs shortest path (APSP) problem, which has the same input as the SSSP, except that no source vertex is given, and the goal is to find for each pair (u, v) of vertices a shortest path connecting them.

A basic observation is that sub-paths of shortest paths are shortest paths again. Hence a shortest path connecting u and v can be obtained from appending the edge (x, v), where x is a neighbor of v, to a shortest path from u to x. This allows a very natural DP formulation as described for problem P.

For APSP, the search space $\mathcal{S}$ naturally is the set of all paths in $\mathcal{G}$, and the set $\mathcal{D}$ of feasible solutions consists of collections of paths, where for each pair of vertices there is one path connecting them.

We model paths via finite sequences of vertices, and do not allow cycles. Since adding a vertex to a path may create a sequence of vertices which does not correspond to a path in $\mathcal{G}$, let us extend this search space to the set $\mathcal{S}'$ of all sequences of vertices of length at most n. The set $\mathcal{S}_0$ of initial solutions is the set of all paths of length 0, that is, of all sequences consisting of a single vertex. Now for all i, we define $\mathcal{F}_i := \{F_v \mid v \in V\} \cup \{\mathrm{id}\}$, where $F_v \colon \mathcal{S} \to \mathcal{S}'$ is the mapping adding the vertex v to a sequence of vertices. To exclude invalid solutions, let us define $H_i(S)$ to be -1 if S is a path in $\mathcal{G}$, and 1 if not.

It remains to define when one state dominates another. Let π_{ij} denote the set of all paths starting in vertex i and ending in vertex j. Let the vector objective function $g \colon \mathcal{S}' \to (\mathbb{R} \cup \{\infty\})^d$, $d = n^2$ have components $g_{ij}(S)$ for all $i, j \in V$, equal to the length of path S iff $S \in \pi_{ij}$. For all other $S \notin \pi_{ij}$ assume $g_{ij}(S) = \infty$. This implies that $S \preceq_{\mathrm{dom}} S'$ if and only if the paths S and S' connect the same two vertices and S' is not longer than S.

Since the length of the path arising from extending an existing path by an edge depends monotonically on the length of the existing path, Conditions 2.1 and 2.2 hold. So, in view of Proposition 2.3, any set $\mathcal{T}_i$ contains a path for each pair of vertices (and only one such path). Thus, $\mathcal{T}_n$ is a subset of $\mathcal{D}$ and contains a shortest path for any pair of vertices.

The resulting algorithm following the dynamic programming approach now does the following. It starts with all paths of length zero as solution set $\mathcal{S}_0$. It then repeats n times the following. For each path in the solution set and each vertex, it appends the vertex to the path. If the resulting path dominates an existing solution with the same end vertices, it replaces the latter. Here $\theta_F = \Theta(1)$, $\theta_{\mathcal{H}} = \Theta(1)$, $\theta_{\preceq} = \Theta(1)$ and $|\mathcal{T}_i| = O(n^2)$, $|\mathcal{F}_i| = O(n)$ for all $i = 1, \ldots, n$, thus (2.4) gives the time complexity bound $O(n^4)$. Note that the well-known Floyd-Warshall algorithm (see e.g. [23], Chapter 25) has $O(n^3)$ time bound, but in that algorithm each transition mapping combines two states (paths), and such an option is not considered here.

2.4 Theoretical Results for EAs

In the following, we show how results of dynamic programming can be attained by evolutionary algorithms. To this aim, we state a general formulation of such an evolutionary algorithm and then describe how the different components have to be designed.

2.4.1 Framework for Evolutionary Algorithms

An evolutionary algorithm consists of different generic modules, which have to be made precise by the user to best fit the problem. Experimental practice, but also some theoretical work (see e. g. [31, 35, 36, 109]), demonstrate that the right choice of representation, variation operators, and selection method is crucial for the success of such algorithms.

We assume again that an instance of problem P is given by a multi-objective function g that has to be optimized. We consider simple evolutionary algorithms that consist of the following components.

We use $\mathcal{S}'_{\mathrm{EA}} := \{0, \ldots, n\} \times \mathcal{S}'$ as the *phenotype space* and call its elements *individuals*. The algorithm (see Algorithm 2.2) starts with an initial population of individuals $\mathcal{P}_0$. During the optimization the evolutionary algorithm uses a selection operator $\mathrm{sel}(\cdot)$ and a mutation operator $\mathrm{mut}(\cdot)$ to create new individuals. The d-dimensional objective function together with a partial order $\preceq$ on $\mathbb{R}^d$ induce a partial quasi-order $\preceq_{\mathrm{EA}}$ on the phenotype space, which guides the search. After the termination of the EA, an output function $\mathrm{out}_{\mathrm{EA}}(\cdot)$ is utilized to map the individuals in the last population to search points from the DP search space.

Algorithm 2.2: Evolutionary Algorithm for P

```
1 P ← ∅;
2 for I ∈ P_0 do
3     if ∄I' ∈ P: I ≺_EA I' then
4         P ← (P \ {I' ∈ P | I' ≺_EA I}) ∪ {I};
5 while true do
6     I ← mut(sel(P));
7     if ∄I' ∈ P: I ≺_EA I' then
8         P ← (P \ {I' ∈ P | I' ≺_EA I}) ∪ {I};
9 return {out_EA(I) | I = (i, S) ∈ P, S ∈ D};
```

2.4.2 Defining the Modules

We now consider how the different modules of the evolutionary algorithm have to be implemented so that it can carry out dynamic programming. To do this, we relate the modules to the different components of a DP algorithm. Consider a problem P given by a set of feasible solutions $\mathcal{D}$ and a multi-objective function g that can be solved by a dynamic programming approach. The EA works with the following setting.

The initial population is $\mathcal{P}_0 = \{0\} \times \mathcal{S}_0$ where $\mathcal{S}_0$ is the initial state space of the DP algorithm. The selection operator $\mathrm{sel}(\cdot)$ chooses an individual $I \in \mathcal{P}$ the following way. First it chooses $i \leq n-1$ uniformly from the set of phases which are represented in the current population i. e. from the set $\{k : k \leq n-1,\ \exists (k,S) \in \mathcal{P}\}$. After this, selection chooses I uniformly among the individuals of the form (i,S) in the current population.

For an individual (i,S), the mutation operator $\mathrm{mut}(\cdot)$ chooses a state transition function $F \in \mathcal{F}_{i+1}$ uniformly at random and sets $\mathrm{mut}((i,S)) = (i+1, F(S))$.

We incorporate a partial order $\preceq_{\mathrm{EA}}$ into the EA to guide the search. This relation is defined as follows:

$$(i,S) \preceq_{\mathrm{EA}} (i',S') \Leftrightarrow (i = i' \text{ and } S \preceq_{\mathrm{dom}} S') \text{ or } H_i(S) > 0. \tag{2.6}$$

Finally, we utilize the output function $\mathrm{out}_{\mathrm{EA}}((i,S)) = S$ to remove the additional information at the end of a run of the EA. That is, we remove the information that was used to store the number of a certain round of the underlying dynamic program and transform an individual into a search point for the problem P.

Note that the description of the Algorithm 2.2 does not employ the notion of the fitness function, although an appropriate multi-objective fitness function may be defined for compatibility with the standard EA terminology.

A Note on the Genetic Mechanisms Corresponding to our Mutation

Let us discuss the solution encoding and the mutation operator which at first glimpse might seem to be well-adapted for the problem at hand. We remark that there actually is a biological analogy to the genetic mechanisms corresponding to the proposed mutation. Here each of the genes A_i, $i = 1, \ldots, n$ defines the DP transition mapping from a set $\mathcal{F}_i$, and a composition of these mappings yields the DP state represented by the individual. One of

the possible options of each gene is "undefined", and the mutation operator modifies the first gene which is still "undefined" in the parent individual.

This is a special case of the point mutation, where a gene A_i subject to change is selected as the first gene which has never been mutated so far (i. e. the first "undefined" gene). Such type of mutation may be imagined in a biological system as follows.

Suppose that for each phase i, $i = 1, \ldots, n$, there is a "controlling" gene B_i. The required localization of mutations in gene A_i, when A_i is the first "undefined" gene, is caused by insertion of some mobile DNA sequence C_i (e.g. a transposon, see [141]), that can enter the locus of gene A_i, and only this locus. We can additionally assume that a mobile element C_i is produced if and only if the gene B_i is active (i. e. B_i is subject to transcription in the parent individual). Besides that, we can assume that gene A_i in the "undefined" condition is silencing the transcription of gene B_{i+1}, but any mutated state of gene A_i activates the transcription of gene B_{i+1} and silences the gene B_i.

Then one can assume that in the i-th generation, $i = 1, \ldots, n$, only the gene B_i is active among $B_1, \ldots, B_n$, provided that initially only the gene B_1 was active. At the same time, in the i-th generation, $i = 1, \ldots, n$, the insertion mutations occur only in the gene A_i.

In nature, an example of a mutually exclusive genes activation is observed in malaria parasite Plasmodium falciparum. The transitions from one variant of a gene to another one depend on the currently active gene variant [77].

2.4.3 Runtime of the Evolutionary Algorithm

Our goal is to show that the evolutionary algorithm solves the problem P efficiently if the dynamic programming approach does. To measure the time the evolutionary algorithm needs to compute a Pareto-optimal set for problem P, one would analyze the expected number of fitness evaluations to come up with a Pareto-optimal set, when it is non-empty. This is also called the expected *optimization time*, which is a common measure for analyzing the runtime behavior of evolutionary algorithms. The proposed EA does not use the multi-objective fitness function explicitly, but given enough memory, it may be implemented so that every individual constructed and evaluated in Lines 3 and 4 or in Lines 6–8 of Algorithm 2.2 requires at most one evaluation of the objective function g. Thus, we can define the optimization time for Algorithm 2.2 as $|\mathcal{S}_0|$ plus the number of iterations of the main loop (Lines 6–8) required to come up with a Pareto-optimal set. Analogous parameter of a DP algorithm is the number of states computed during its execution.

The next theorem relates the expected optimization time of the EA to the number of states computed during the execution of the corresponding DP algorithm. In what follows it will be convenient to denote the cardinality of the set of states produced after completion of the DP algorithm by $|DP|$, i. e. $|DP| := \sum_{i=0}^{n} |\mathcal{T}_i|$. Note that $|DP|$ is a well-defined value since the sizes $|\mathcal{T}_i|$ are unique according to Proposition 2.4.

Theorem 2.5. *Let a DP be defined as in Algorithm 2.1 and an EA defined as in Algorithm 2.2 with $\preceq_{\text{EA}}$ relation defined by (2.6). Then the number of states computed during the execution of the DP algorithm is $|\mathcal{S}_0| + \sum_{i=1}^{n} |\mathcal{F}_i| \cdot |\mathcal{T}_{i-1}|$, and the EA has an expected optimization time of*

$$O\left(|\mathcal{S}_0| + n \cdot \log |DP| \cdot \sum_{i=0}^{n-1} |\mathcal{T}_i| \cdot |\mathcal{F}_{i+1}|\right).$$

Proof. Estimation of the number of states computed during the execution of the DP algorithm is straightforward.

Assume that the optimization process works in stages $1 \leq i \leq n$, whereas stage $i+1$ starts after the stage i has been finished. We define that a stage i finishes when for every state $S \in \mathcal{T}_i$ there exists an individual $(i, S') \in \mathcal{P}$ with S' dominating S. Here and below in this proof, by $\mathcal{T}_i,\ i = 0, \ldots, n$, we denote the corresponding sets computed in Algorithm 2.1. Note that after completion of a stage i, the subset of individuals of a form (i, S) in population $\mathcal{P}$ does not change in the subsequent iterations of the EA. Let $\mathcal{T}'_i$ denote the set of states of these individuals after completion of stage $i,\ i = 0, \ldots, n$. By the definition of Algorithm 2.2, the sequence $\mathcal{T}'_i,\ i = 0, \ldots, n$ satisfies (2.2), and therefore $|\mathcal{T}'_i| = |\mathcal{T}_i|,\ i = 0, \ldots, n$ in view of Proposition 2.4.

Let ξ_i be the random variable denoting the number of iterations since stage $i-1$ is finished, until stage i is completed. Then the expected optimization time is given by $|\mathcal{S}_0| + E[\xi]$ with $\xi = \xi_1 + \cdots + \xi_n$.

Any state $S \in \mathcal{T}_{i+1}$ is computed in Algorithm 2.1 by means of some function $\tilde{F} \in \mathcal{F}_{i+1}$, when it is applied to some state $\tilde{S} \in \mathcal{T}_i$. Thus, in stage $i+1$ of the EA during mutation the same transition function $\tilde{F}$ may be applied to some individual $I' = (i, S')$, such that $\tilde{S} \preceq_{\text{dom}} S'$. After this mutation, in view of Conditions 2.1 and 2.2, the population $\mathcal{P}$ will contain an individual $I'' = (i+1, S'')$ with S'' such that $S \preceq_{\text{dom}} \tilde{F}(S') \preceq_{\text{dom}} S''$.

Consider any iteration of the EA at stage $i+1$. Let t denote the number of such states from $\mathcal{T}_{i+1}$ that are already dominated by a state of some individual

in $\mathcal{P}$. Then there should be $|\mathcal{T}_{i+1}| - t$ new individuals of the form $(i+1, S)$ to be added into $\mathcal{P}$ to complete stage $i+1$ (recall that $|\mathcal{T'}_{i+1}| = |\mathcal{T}_{i+1}|$). The probability to produce an individual (i, S') where S' dominates a previously non-dominated state from $\mathcal{T}_{i+1}$ is no less than $(|\mathcal{T}_{i+1}| - t)/(n|\mathcal{T}_i| \cdot |\mathcal{F}_{i+1}|)$ with an expected waiting time of at most $(n|\mathcal{T}_i| \cdot |\mathcal{F}_{i+1}|)/(|\mathcal{T}_{i+1}| - t)$ for this geometrically distributed variable. The expected waiting time to finish stage $i+1$ is thus bounded by

$$E[\xi_{i+1}] \leq \sum_{t=1}^{|\mathcal{T}_{i+1}|} \frac{n|\mathcal{T}_i| \cdot |\mathcal{F}_{i+1}|}{t} = n|\mathcal{T}_i| \cdot |\mathcal{F}_{i+1}| \cdot \mathcal{H}_{|\mathcal{T}_{i+1}|},$$

with $\mathcal{H}_k$ being the k-th harmonic number, $\mathcal{H}_k := \sum_{i=1}^{k} \frac{1}{i}$.

This leads to an overall expected number of iterations

$$E[\xi] \leq \sum_{i=0}^{n-1} n|\mathcal{T}_i| \cdot |\mathcal{F}_{i+1}| \cdot \mathcal{H}_{|\mathcal{T}_{i+1}|} \leq n(\ln |DP| + 1) \cdot \sum_{i=0}^{n-1} |\mathcal{T}_i| \cdot |\mathcal{F}_{i+1}|. \qquad \square$$

A similar inspection as in Subsection 2.2.2 reveals that the expected runtime of the EA is

$$O\left(\theta_{\text{ini}} + n \log |DP| \cdot \sum_{i=0}^{n-1} \left(|\mathcal{F}_{i+1}| \cdot |\mathcal{T}_i| \cdot (\theta_{\mathcal{F}_i} + \theta_{\mathcal{H}} + \theta_{\preceq})\right) + \theta_{\text{out}}\right),$$

assuming the individuals of the population are stored in $n+1$ disjoint sets according to the first coordinate i.

As noted in Subsection 2.2.2, if the computation times for functions F, H_i and dominance checking (2.3) as well as execution time for Line 10 in Algorithm 2.1 are constant, then θ_F, $\theta_{\mathcal{H}}$ and $\theta_{\preceq}$ can be chosen *equal* to the corresponding computation times. In such cases a problem that is solved by dynamic programming Algorithm 2.1 in time T, will be solved by the EA defined as in Algorithm 2.2 in expected time $O(Tn \log |DP|)$.

2.4.4 Homogeneous Transitions

Some DP algorithms, like the ones for APSP and SSSP, have a specific structure which may be exploited in the EA. In this subsection we consider the case of *homogeneous* transition functions where $\mathcal{F}_1 \equiv \cdots \equiv \mathcal{F}_n$ and $H_1 \equiv \cdots \equiv H_n$. To simplify the notation in this case we will assume $\mathcal{F}_1 \equiv \mathcal{F}$ and $H_1(S) \equiv H(S)$. Additionally, we suppose that the identical mapping belongs to $\mathcal{F}$.

The formulated assumptions imply that once some state S is obtained in the DP algorithm, it will be copied from one phase to another, unless some other state will dominate it. Note also that it does not matter at what particular phase a state has been obtained – the transition functions will produce the same images of this state. These observations motivate a modification of the partial order $\preceq_{\mathrm{EA}}$, neglecting the phase number in comparison of individuals:

$$(i, S) \preceq_{\mathrm{EA}} (i', S') \Leftrightarrow S \preceq_{\mathrm{dom}} S' \text{ or } H_i(S) > 0. \tag{2.7}$$

In fact, now we can skip the index i in individuals (i, S) of the EA, so in this subsection the terms "state" and "individual" are synonyms and the phase number i is suppressed in the notation of individuals. As the following theorem shows, wider sets of comparable individuals in this special case allow to reduce the population size and thus improve the performance of the EA. Let us consider the *width* W_{dom} of partial order $\preceq_{\mathrm{dom}}$, i. e. the maximum size of a set of pairwise incomparable elements.

Theorem 2.6. *If the transition functions are homogeneous and* $id \in \mathcal{F}$*, then the EA defined as in Algorithm 2.2 with the modified* $\preceq_{\mathrm{EA}}$ *relation (2.7) has an expected optimization time of* $O(|\mathcal{S}_0| + W_{\mathrm{dom}} \log(W_{\mathrm{dom}}) \cdot n|\mathcal{F}|)$.

Proof. The analysis is similar to the proof of Theorem 2.5. Note that now the size of population $\mathcal{P}$ does not exceed W_{dom}. We assume that $|\mathcal{P}| = W_{\mathrm{dom}}$ right from the start.

Let $\mathcal{T}_i$ be the same as in phase i of the DP algorithm, $i = 0, \ldots, n$. Suppose again that the optimization process works in stages $1 \le i \le n$, whereas stage i is assumed to be finished when for every $S \in \mathcal{T}_i$, the population $\mathcal{P}$ contains an individual S' such that $S \preceq_{\mathrm{dom}} S'$.

Let ξ_i be the number of iterations since stage $i-1$ is finished, until stage i is completed. Then the expected optimization time is given by $|\mathcal{S}_0| + E[\xi]$ with $\xi = \xi_1 + \cdots + \xi_n$.

Any state $S \in \mathcal{T}_{i+1}$ is computed in the DP algorithm by means of some function $\tilde{F} \in \mathcal{F}_{i+1}$, when it is applied to some state $\tilde{S} \in \mathcal{T}_i$. Thus, in stage $i+1$ of the EA during mutation the same transition function $\tilde{F}$ may be applied to some individual $I' = S'$, such that $\tilde{S} \preceq_{\mathrm{dom}} S'$. After this mutation, in view of Conditions 2.1 and 2.2, the population $\mathcal{P}$ will contain an individual $I'' = S''$ such that $S \preceq_{\mathrm{dom}} \tilde{F}(S') \preceq_{\mathrm{dom}} S''$.

The probability of such a mutation for a particular $S \in \mathcal{T}_{i+1}$ is at least $1/(|\mathcal{F}_{i+1}| \cdot |\mathcal{P}|) \leq 1/(|\mathcal{F}_{i+1}| \cdot W_{\text{dom}})$. Let t denote the number of states $S \in \mathcal{T}_{i+1}$ that are already dominated at stage $i+1$. Then there are at least $|\mathcal{T}_{i+1}| - t$ possibilities to add a new individual, which dominates a previously non-dominated state from $\mathcal{T}_{i+1}$. The probability for such a mutation is not less than $(|\mathcal{T}_{i+1}| - t)/(W_{\text{dom}} \cdot |\mathcal{F}_{i+1}|)$ with an expected waiting time of at most $(W_{\text{dom}} \cdot |\mathcal{F}_{i+1}|)/(|\mathcal{T}_{i+1}| - t)$ for this geometrically distributed variable. The expected waiting time to finish stage $i+1$ is thus $E[\xi_{i+1}] \leq W_{\text{dom}} \cdot |\mathcal{F}_{i+1}| \cdot \mathcal{H}_{|\mathcal{T}_{i+1}|}$. But $|\mathcal{T}_{i+1}| \leq W_{\text{dom}}$ because the states of $\mathcal{T}_{i+1}$ are pairwise incomparable according to Algorithm 2.1. This leads to an overall expected number of iterations $E[\xi] \leq W_{\text{dom}} \cdot (\ln(W_{\text{dom}}) + 1) \cdot \sum_{i=0}^{n-1} |\mathcal{F}_{i+1}|$. □

2.4.5 Examples

Now, we point out how the framework presented in this section can be used to construct evolutionary algorithms using the examples from Section 1.3.

Traveling Salesman Problem. Due to Theorem 2.5 the expected optimization time of the evolutionary algorithm based on the DP algorithm of Held and Karp presented in Section 2.3 is $O\big(n^4 2^n\big)$. This bound can be further improved to $O\big(n^3 \cdot 2^n\big)$ for the EA proposed in [150].

Knapsack Problem. Consider the DP algorithm presented in Section 2.3. The expected optimization time of the corresponding EA for the knapsack problem is $O\big(n^2 \cdot W \cdot \log(n \cdot W)\big)$ due to Theorem 2.5.

Single Source Shortest Path Problem. Application of Theorem 2.5 to the DP algorithm for SSSP problem from Section 2.3 gives an expected optimization time of $O\big(n^4 \log(n)\big)$ for Algorithm 2.2.

The DP algorithm for SSSP problem has homogeneous transition functions with $W_{\text{dom}} = n$. Thus, the modified EA considered in Theorem 2.6 has the expected optimization time $O(n^3 \log n)$. This bound can be further improved to $O(n^3)$ for the (1+1) EA [138].

All-Pairs Shortest Path Problem. Plugging the ideas of the DP algorithm for APSP presented in Section 2.3 into the framework of Algorithm 2.2, we obtain an EA with an expected optimization time of $O\big(n^5 \log(n)\big)$ due to Theorem 2.5.

It has been noted, however, that the DP algorithm for APSP has homogeneous transition functions, each set $\mathcal{F}_i$ contains the identical mapping. Here $W_{\text{dom}} = n^2$, thus Theorem 2.6 implies that the modified EA has the expected optimization time $O(n^4 \log n)$. This algorithm can be further improved to an EA with optimization time $\Theta(n^4)$ as has been shown in [44].

2.5 Experiments for TSP

In this section we discuss some otherwise unpublished experiments that were undertaken to check the hypotheses that crossover is a useful extension for an evolutionary algorithm with DP-capabilities. To this end we chose the Traveling Salesperson Problem, which is NP-hard and inapproximable within a constant factor in the general case under the assumption P≠NP, see [139, Chapter 58] for a comprehensive introduction. This rules out a polynomial time algorithm for the exact or even approximate solution of the problem. For the general case the only approach known to deliver exact and optimal solutions is the exponential-time algorithm of Held and Karp discussed in this chapter. Nowadays, there also exist many successful heuristics, which allow to treat TSP instances of realistic size. However, we do not want to compete with them. The goal in this section is to evaluate the assumption that crossover helps with the optimization of EAs with the ability to carry out dynamic programming.

Here, we use the specific representation described in Section 2.3 as well as in the authors single-authored publication [150]. An individual $\pi(k, M)$ is a Hamiltonian cycle on a ground set $M \subseteq V \setminus \{1\}$ that starts in vertex 1 runs through all vertices of M in a fixed order with $k \in M$ being the last vertex of the corresponding Hamiltonian path (cf. Section 2.3). Considering all such possible sets, the size μ of the population is upper bounded by $\mu \leq (n-1) \cdot 2^{n-2}$ for any input graph $G = (V, E)$.

For our experiments we choose a complete, n-vertex graph $G = (V, E)$ with $|V| = n$ with a Hamiltonian cycle implanted in the graph. All edges of the Hamiltonian cycle have weight 1, all other edges weight $N \gg n$. Our experiments generally could not go beyond $n = 17$ within a reasonable iteration limit described below for each experiment. These rather small sizes of n are justified because the population size and thus the runtime of the $(\mu + 1)$ GA increases exponentially with n (compare the theoretical bound on the optimization time in Section 2.3). Here, optimization time denotes the average number of iterations measured within a set of experiments.

Algorithm 2.3: $(\mu + 1)$ GA for TSP

Initialize the population $\mathcal{P}$ with μ different individuals $x_1, \ldots, x_\mu$ with $x_i \leftarrow \pi(k, M)$ for all combinations of $M \subseteq V \setminus \{1\}$ and $k \in M$;
while *true* **do**
  Choose $c \in [0, 1]$ uniformly at random;
  if $c \leq p_c$ **then**
    Choose two individuals z' and z'' from $\mathcal{P}$ uniformly at random;
    Perform CROSSOVER on z' and z'' to obtain an individual z;
  else
    Select individual $z' \leftarrow \pi(k', M') \in \mathbb{P}$ uniformly at random and perform MUTATION;
    Choose $s \leftarrow \text{Pois}(\lambda = 1)$;
    Generate new individual $z \leftarrow \pi(k, M)$ by $s + 1$ times applying the mutation operator to z;
  Perform ENVIRONMENTAL SELECTION;
  Let $\hat{z} \leftarrow \pi(k, M) \in \mathcal{P}$ be the individual defined on the same ground set having the same end vertex if such an individual exists in the population;
  if $f(z) \leq f(\hat{z})$ **then**
    Add z to $\mathcal{P}$ and remove $\hat{z}$ from $\mathcal{P}$;

We employ the $(\mu{+}1)$ GA given in Algorithm 2.3, which uses a crossover operator with probability $p_c \in [0, 1]$ and a mutation operator with probability $1 - p_c$. A single step of the *mutation operator* applied to individual $\pi(k, M)$ works as follows: A node q is chosen uniformly at random from the set $V \setminus \{1\} \setminus M$. The edge (k, q) is appended at the end of the Hamiltonian path underlying $\pi(k, M)$. Note, that the operator only produces feasible individuals. We refer the reader to page 62 of Section 4.1 for a detailed discussion of this specific mutation operator. Our basic variant of the *crossover operator* chooses two individuals uniformly at random from the population and appends the second to the first individual. As it is very likely that both ground sets have a vertex in common we apply a repair mechanism. It is not reasonable to allow the creation of infeasible individuals arising from recombining two ground sets into an offspring if they have a vertex in common. Experimentally this intuition leads to higher optimization times. Our *repair mechanism* excludes vertices from the ground set of the second individual if this vertex is already contained in the first individual. Note, that this mechanism keeps the ordering of the remaining vertices in the second individual fixed.

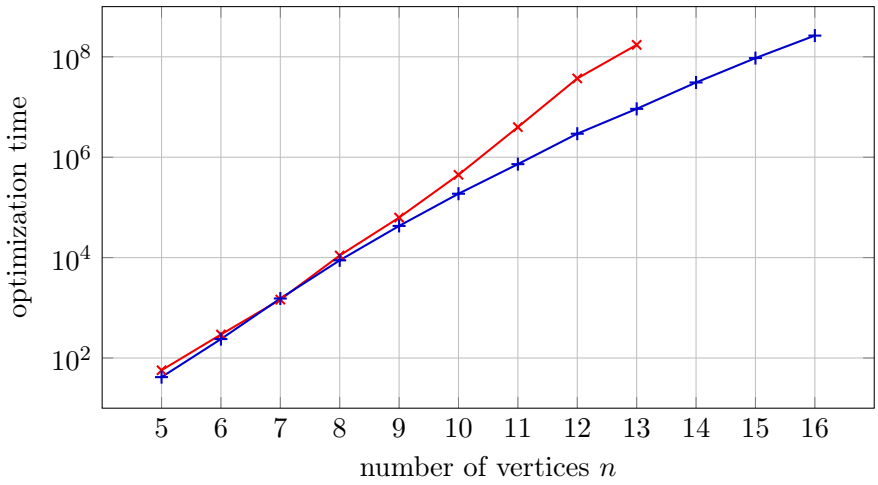


Figure 2.1: Comparison of the empirical optimization times for the $(\mu+1)$ GA with $p_c := 1/2$ using the basic crossover operator (—×—) and the $(\mu+1)$ EA using mutation only (—+—) on logarithmic scales.

Basic Crossover. We call the aforementioned simple repair crossover operator our *basic crossover*.

Cut First Crossover. The first variant of the basic crossover operator chooses two individuals uniformly at random from the population. The first of both individuals $\pi(k_1, M_1)$ is cut off at position p_1 for a number $p_1 \in \{2, \dots |M_1|\}$ chosen uniformly at random. The second individual is then appended to the first individual using the repair mechanism described above. We call this operator *cut first crossover*.

Cut Both Crossover. The second variant which we call *cut both crossover* again chooses two individuals uniformly at random from the population and cuts both individuals before combining them. Let $\pi(k_1, M_1)$ be the first and $\pi(k_2, M_2)$ be the second individual. For each individual a number $k_1 \in \{2, \dots, |M_1|\}$ and $k_2 \in \{2, \dots, |M_2|\}$, respectively, is chosen uniformly at random. The second individual which is cut off at k_2 is appended to the first individual which is cut off at position k_1 using the well-known repair mechanism.

The aim of the *environmental selection* is to prevent the population from growing too large while at the same time the "right" individuals have to remain in the population (cf. line 12 and 13 in Algorithm 2.3). In our case it acts as a diversity mechanism to ensure that each possible individual is contained in the population at most once. For each permutation $\pi(k, M)$ only the fittest will stay in the population, such that we only compare the fitness of individuals which are identical with respect to their ground set M and vertex k.

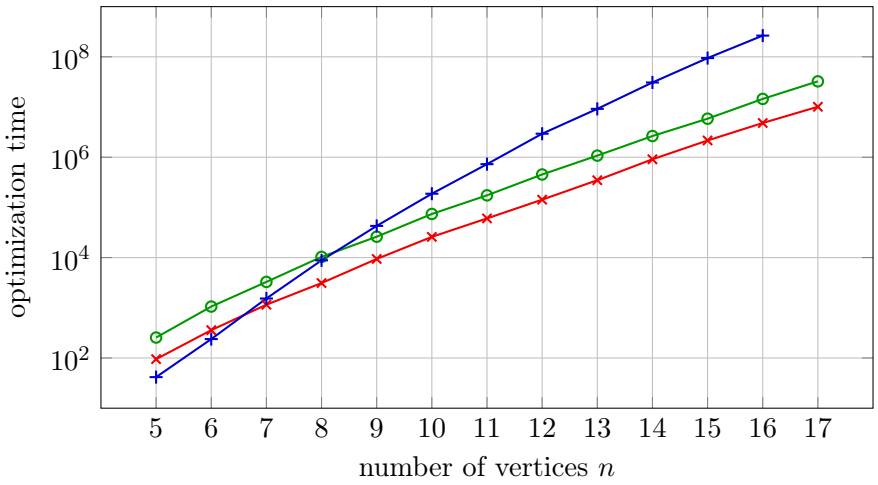


Figure 2.2: Comparison of the empirical optimization times for the two variants of the basic crossover operator. The result of the cut both crossover (—o—) and the cut first crossover (—×—) is shown in comparison to mutation only (—+—) on logarithmic scales.

In Figure 2.1 we compare the effect of the $(\mu+1)$ GA with $p_c := 1/2$ employing the basic crossover to a $(\mu + 1)$ EA with $p_c := 0$ using mutation only. For each $n \in \{5, \ldots, 13\}$ for the $(\mu + 1)$ GA and for each $n \in \{5, \ldots, 16\}$ for the $(\mu + 1)$ EA we ran at least 50 tests. Interestingly the $(\mu + 1)$ GA using the basic crossover operator (—×—) does not outperform the $(\mu + 1)$ EA (—+—). In fact, its optimization time increases so drastically that an iteration limit of $1.4 \cdot 10^9$ does not suffice to go beyond $n = 13$.

However, employing the two variants of the basic crossover already results in a visible increase in performance depicted in Figure 2.2. For our experiments we used at least 50 repetitions for each $n = \{5, \ldots, 17\}$. We would have expected that the increased flexibility of the cut both crossover (—o—) operator results in an even better optimization time compared to the cut first crossover (—×—). Experimentally this could not be confirmed. It is not clear whether the reason lies in the repair mechanism or whether it is just the simple argument that cutting both individuals needs two lucky events ($1/n^2$) instead of just a single one ($1/n$). In any case, for this instance the comparison of the basic crossover in Figure 2.1 and the cut first end crossover in Figure 2.2 show it to be advantageous to add some but not too much flexibility to the crossover operator.

The large population size is an obvious drawback of the dynamic programming representation. Due to this we ran a set of experiments for mutation only, i. e. $p_c := 0$, in which we restricted the population size. Dividing the population into disjoint layers according to the size $|M| = 1 \ldots n - 1$ of the ground set,

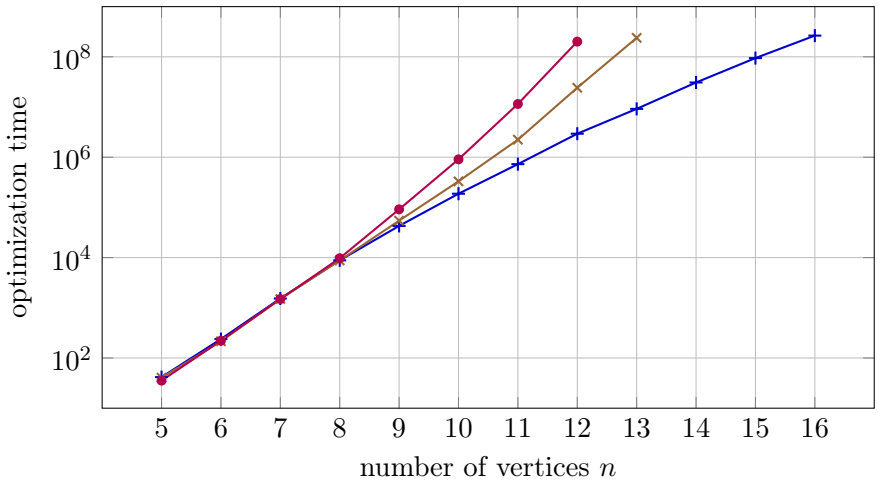


Figure 2.3: Comparison of the empirical optimization times for mutation only using the full population size (—+—) and mutation with only half the number of individuals per layer allowed (—×—) and mutation with only an n-th of the individuals per layer allowed (—•—) on logarithmic scales.

we restricted the size of each layer to a fraction of $1/2$ and $1/n$ of the maximum number of individuals per layer. If an individual is to be inserted into the population in Line 14 of Algorithm 2.3 and the size bound of its layer is reached then we remove an individual from the population. We choose to remove an individual uniformly at random from the full layer, because we cannot expect to gain any information for the removal from the fitness of an individual. Thus, it comes as no surprise that our experiments depicted in Figure 2.3 suggest that the approach to restrict the population size is not reasonable compared to actually storing the full population.

Each run for $n = \{5, \dots, 16\}$ was repeated at least 50 times with a varying iteration limit. All runs for the full population size (—+—) were completed within a $3 \cdot 10^9$ iterations. The same time limit for our experiments with half the population size (—×—) only allowed to go up to $n = 13$ for which only 98% of the problems could be solved. For larger instances $n > 13$ less than 80% were solved such that they were omitted from the analysis. The same holds true for our experiments where we only kept a fraction of $1/n$ (—•—) for each layer. Here, it was hardly possible to solve instances larger $n > 12$ within the generous timebound of 10^9 iterations.

2.5.1 Conclusion

With our experiments we tackled the question whether it is possible to decrease the optimization time of the $(\mu + 1)$ EA for TSP based on the DP-representation described in Section 2.3. Experimentally we compared the optimization times of the $(\mu+1)$ EA to our basic crossover operator. It seems to be reasonable that the basic crossover fails to give an improvement because it is more likely to choose two too long individuals for recombination in later stages of the optimization process. Either the recombination fails because the symmetric difference of the ground sets is empty or the increase in the length of the offspring can easily be made up by mutation only.

Adding just some flexibility to the basic crossover operator shaves off a significant factor of the optimization time using the cut first crossover. However, even more flexibility, for example like in the cut both crossover, did not result in yet another decrease in the optimization time. It would have to be investigated whether this is due to the repair mechanism or whether it is because we cannot easily hope for two lucky events choosing two good cut off points at the same time.

When we first started this set of experiments, we had in mind that it must be possible to gain a dramatic increase in performance for TSP. Thus, we took an approach into the direction to reduce the large population size. However, even for the $(\mu + 1)$ EA the results are shattering as shown in Figure 2.3. Even for this easy instance of TSP the $(\mu+1)$ EA is not capable of building an optimal solution from the partial solutions contained in the population. It is clearly problematic to remove an individual from a full layer uniformly at random because we might need this later on. Here, different environmental selections could be helpful, but it is not clear whether the fitness is a reasonable measure to achieve this.

In Chapter 4 we introduce and analyze a genetic algorithm for the all-pairs shortest path problem, which is another problem with a DP-structure discussed in Section 2.3. For this problem we are actually able to prove a speed-up of the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ as compared to a mutation only approach. After having looked very closely into the structure of the problem in Section 4.8 we discuss why it is not easily possible to apply this approach to problems with a DP structure.

3 Recombination in Pseudo-Boolean Optimization

This chapter is based on publication [90], which was co-authored by Timo Kötzing and Dirk Sudholt and was awarded a best paper award of the track Genetic Algorithms at the Genetic and Evolutionary Computation Conference (GECCO) 2011.

In this chapter we present new insights on working principles of crossover by analyzing the performance of crossover-based GAs on the simple Pseudo-Boolean functions OneMax and Jump (already discussed in the introduction).

3.1 Contribution

It is well-known that crossover heavily relies on diversity in the population. Due to this in Section 3.2 we abstract from the influence of the diversity and consider a setting for uniform crossover with a rather ideal diversity. This diversity is obtained by combining crossover with a specialized but *fitness-invariant* operator that is not found in typical GAs. The idea is to randomly shuffle all bits in the bit string before performing a uniform crossover. This does not affect the fitness on functions of unitation (i. e. functions depending on the number of ones only), but it simulates the potential diversity if we had run two independent strands of evolution on a unitation function. As such, it can create the diversity necessary to effectively perform crossover. Note

that this crossover, called *shuffle crossover*, is not a geometric crossover as defined in [107] and discussed in the introduction. The purpose is to assess the potential speedup of crossover in the presence of ideal diversity. We prove that this can lower the expected running time drastically—compared to the (1+1) EA without crossover—on OneMax from $\Theta(n \log n)$ to $\Theta(\sqrt{n})$ function evaluations and on Jump_k from $\Theta(n^k)$ to $\Theta(\sqrt{n} + 4^k)$. This illuminates the possible advantage of crossover compared to mutation-only, depending on diversity.

More realistic GAs without shuffling usually obtain diversity by having large populations. The GA considered by Jansen and Wegener [86] needs a quite large population and a very low crossover probability to be effective on Jump_k. In Section 3.3 we investigate the interplay of mutation and crossover on Jump_k. We show that large populations are not necessary to provide the required diversity if the crossover probability is small. This holds even up to population sizes as small as 2. On the other hand, large crossover probabilities make small populations collapse quickly, rendering crossover inefficient.

Thereby, we discuss an interesting effect. With standard, geometric crossovers EAs perform a convex search on the population, gradually decreasing the convex hull spanned by all members of the population until it eventually collapses to a single point [107, Section 3.5]. Contrarily, mutation can create new points outside the present convex hull and thus randomly extends the convex hull. The interplay of these conflicting operators leads to an equilibrium state. The diversity in such an equilibrium state is essential for optimization. We provide Monte Carlo simulations in Section 3.4 in the regime not covered by our theoretical results to gain deeper insights into this equilibrium and the respective diversity for various parameter settings on the Jump plateau. This leads to novel insights on when crossover can create a global optimum.

3.2 On the Potential of Crossover

In contrast to mutation operators that typically only make small changes, crossover can yield a larger progress and a greater increase in fitness. This, however, only holds if the diversity in the population is large enough: if all individuals in the population are very similar, crossover will create a similar offspring. The analysis of diversity is a challenging problem in its own right [65]. We are interested in the *potential* of crossover. How large can the progress by crossover be if there is sufficient diversity in the population?

To answer this question we consider uniform crossover for a population where there is an "ideal" diversity. It is ideal in a sense that it models the situation where two individuals have been evolved independently to the same fitness value. On functions of unitation the fitness only depends on the number of bits set to 1. If mutation and/or uniform crossover is used (as opposed to 1-point or k-point crossover where linkage of bits comes into play), the position of 1-bits will be uniform if we look at one specific individual in the population.

We consider a shuffling operator that randomly permutes all bits in order to simulate a population that has evolved independently. Introducing superb diversity into the population by this operator allows us to abstract from the influence of other diversity mechanisms. For functions of unitation this operator is reasonable as it does not affect the fitness. It needs to be remarked, though, that this operator is clearly artificial and tailored towards functions of unitation. In particular, results obtained in this setting do not generalize under straightforward transformations of the search space such as exchanging bits values for selected bits. This in particular eludes lower bounds on the black-box complexity for such generalizations [51, 97]. Nevertheless, it allows us to observe the potential progress by crossover in a clear fashion.

The following algorithm called *shuffle GA* uses a population size 1 and the shuffling mechanism to create a second parent out of the current search point. This idea is similar to the gene invariant GA (GIGA) [24, 27].

Algorithm 3.1: Shuffle GA.

```
1 Initialize an individual x uniformly at random;
2 while true do
3     x' ← x;
4     Permute all bits of x' uniformly at random;
5     y ← uniform crossover(x, x');
6     if f(y) ≥ f(x) then x ← y;
```

We abbreviate the two operations shuffling bits and performing a uniform crossover as *shuffle crossover*. Note that, for functions of unitation, it does not matter whether we shuffle one or both parents.

In the remainder of this section we determine tight bounds on the optimization time of the shuffle GA for the functions OneMax and Jump_k. OneMax represents a simple test case where all bits can be seen as small building blocks that have to be assembled. In contrast to the standard (1+1) EA [49] that needs $\Theta(n \log n)$ steps in expectation, uniform crossover can lead to a significant progress. If both parents have an equal number of 1-bits and their

Hamming distance is k, then the gain of the OneMax-value for the offspring is governed by k independent Bernoulli trials. The following lemma shows that there is a good chance of having a surplus of $\Omega(\sqrt{k})$ ones.

Lemma 3.1. *Let X be the sum of independent random variables $X_1, \ldots, X_k \in \{0,1\}$ where $X_i = 1$ with probability $1/2$. Define the surplus of ones as $Y := \max\{0, X - k/2\}$. Then $\mathsf{P}(Y \geq \pi/(4e) \cdot \sqrt{k}) \geq 1/4$ and $\mathrm{E}[Y] \leq \sqrt{k} + O(1)$.*

Proof. Without loss of generality k is even. For every $0 \leq i \leq k$

$$\mathsf{P}(X = i) \leq \mathsf{P}(X = k/2) = \binom{k}{k/2} \cdot 2^{-k} = \frac{k! \cdot 2^{-k}}{((k/2)!)^2}.$$

From Stirling's approximation and a simple case analysis it follows for $k \in \mathbb{N}$ that $\sqrt{2\pi k} \cdot (k/e)^k \leq k! \leq e\sqrt{k} \cdot (k/e)^k$. Hence the above probability is at most

$$\frac{e\sqrt{k} \cdot (k/e)^k \cdot 2^{-k}}{(\sqrt{2\pi \cdot k/2})^2 \cdot (k/(2e))^k} = \frac{e}{\pi\sqrt{k}}.$$

As $\mathsf{P}(X < k/2) = \mathsf{P}(X > k/2) \leq 1/2$, we have that

$$\mathsf{P}\left(X \leq k/2 + \frac{\pi}{4e} \cdot \sqrt{k} - 1\right) \leq \frac{1}{2} + \frac{\pi\sqrt{k}}{4e} \cdot \frac{e}{\pi\sqrt{k}} = \frac{3}{4}.$$

This proves the claimed probability bound for Y.

For the upper bound on $\mathrm{E}[Y]$ we have

$$\mathrm{E}[Y] = \sum_{i=0}^{k/2} \mathsf{P}(Y \geq i) = \sum_{i=0}^{k/2} \mathsf{P}(X \geq k/2 + i).$$

Clearly, $\mathsf{P}(X \geq k) = 2^{-k}$ and $\mathrm{E}[X] = k/2$. For $i < k/2$ we apply well known Chernoff bounds to get

$$\begin{aligned}\mathrm{E}[Y] &= 2^{-k} + \sum_{i=0}^{k/2-1} \mathsf{P}\left(X \geq \left(1 + \frac{2i}{k}\right) \cdot \frac{k}{2}\right) \\ &\leq 2^{-k} + \sum_{i=0}^{k/2-1} e^{-k/6 \cdot (2i/k)^2} = 2^{-k} + \sum_{i=0}^{k/2-1} e^{-2/3 \cdot i^2/k}.\end{aligned}$$

Estimating the first $\sqrt{k}$ summands of the $\sum$-term (without loss of generality assuming $\sqrt{k} \in \mathbb{N}$) by the trivial bound 1 and using $i^2 = (\sqrt{k} + (i - \sqrt{k}))^2 \geq$

$k + (i - \sqrt{k})$ for $i \geq \sqrt{k}$, we arrive at

$$\begin{aligned} &\sqrt{k} + 2^{-k} + \sum_{i=\sqrt{k}}^{k/2-1} e^{-2/3 \cdot (i-\sqrt{k})} \\ &\leq \sqrt{k} + \sum_{i=0}^{\infty} e^{-2/3 \cdot i} = \sqrt{k} + \frac{1}{1 - e^{-2/3}}. \end{aligned}$$ □

The following theorem gives an upper bound on the optimization time of the shuffle GA on OneMax. As the shuffle GA cannot generate 1-bits when starting with 0^n, we have to exclude this case.

> **Theorem 3.2.** *Unless initialized with 0^n, the expected optimization time of the shuffle GA on* OneMax *is $O(\sqrt{n})$.*

Proof. We estimate the increase of the offspring's OneMax-value using the surplus argument of Lemma 3.1. As the crossover operator can only yield a surplus on bits where both parents differ we first derive a lower bound on the number of these bits.

Let $k < n$ be the number of zeros in the current bit string x. First consider the case $k \leq n/2$. Consider the bit string x' obtained by shuffling x. Whenever there is a bit where both x and x' are 0, we speak of a collision. For a fixed 0-bit in x we have that the expected number of collisions of the bit is k/n. By linearity of expectation, the total expected number of collisions is therefore $k \cdot k/n = k^2/n$. Using Markov's inequality, with probability at least $1/3$ we have at most $3/2 \cdot k^2/n$ collisions. This implies that the number of 0-bits in x where x' has value 1 is at least $k - 2k^2/n \geq k/4$ with probability at least $1/3$. The case $k \geq n/2$ follows by symmetric arguments, exchanging the roles of zeros and ones.

Along with Lemma 3.1, the shuffle GA gains at least $\pi/(8e) \cdot \sqrt{k}$ ones with probability at least $1/12$. Other steps cannot decrease the number of ones. If, for the current number of zeros k, it holds that $n/(2^{i+1}) < k \leq n/2^i$ for some $i \in \mathbb{N}$, the expected time until this number has decreased to or below $n/2^{i+1}$ is at most

$$12 \cdot \left\lceil \frac{n}{2^i} \cdot \frac{8e}{\pi \sqrt{n/2^{i+1}}} \right\rceil \leq 12 + \frac{96e}{\sqrt{2}\pi} \cdot \frac{\sqrt{n}}{2^{i/2}}.$$

As we start with at most $n/2^0$ zeros and finish when there are at most $n/2^{(\log n)+1} < 1$ zeros left, summing up all expected time bounds, the total

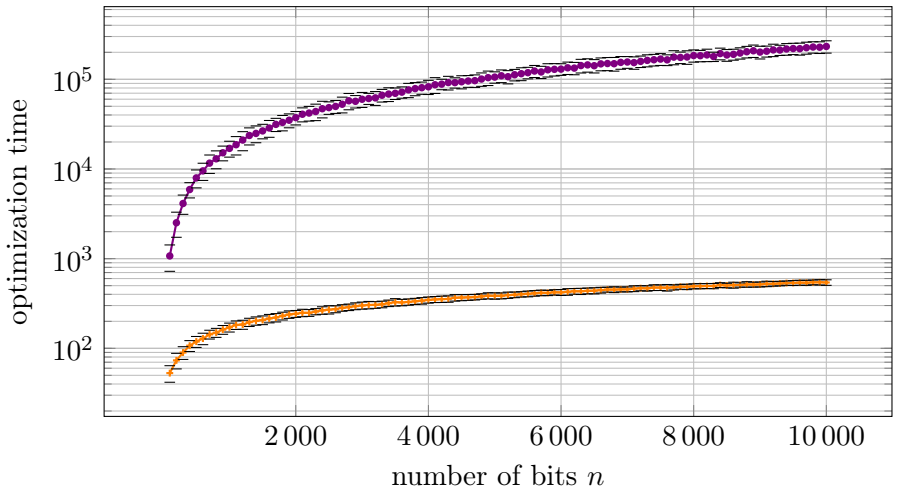


Figure 3.1: Average optimization times and standard deviation (-) for the (1+1) EA (—•—) and the shuffle GA (—•—) on OneMax. Note that the y-axis is scaled logarithmically.

expected optimization time is

$$\sum_{i=0}^{(\log n)+1} \left(12 + \frac{96e}{\sqrt{2}\pi} \cdot \frac{\sqrt{n}}{2^{i/2}}\right)$$
$$\leq O(\log n) + \frac{96e}{\sqrt{2}\pi} \cdot \sqrt{n} \cdot \sum_{i=0}^{\infty} 2^{-i/2} = O(\sqrt{n}). \qquad \square$$

The speedup compared to the time $\Theta(n \log n)$ of the (1+1) EA is dramatic. To see that this is not just a theoretical artifact, we consider experiments for realistic problem dimensions. Figure 3.1 shows average optimization times for $n \in \{100, 200, 300, \ldots, 10\,000\}$ with a hundred runs for each n on a logarithmic scale. The plot shows the huge difference of the runtime between a standard (1+1) EA on OneMax and our shuffle GA. Note that the variance for the shuffle GA is smaller than for the (1+1) EA, i. e. the runtime of the shuffle GA is much more concentrated.

The reason for the overwhelming success of the shuffle GA is that shuffling bits of an individual with i zeros creates another individual uniformly distributed on the Hamming ball around the optimum 1^n. Recombining two parents that are "on opposite sides" of the global optimum can yield a significant progress towards the optimum.

In order to avoid misinterpretations, we stress that Theorem 3.2 does not contradict higher lower bounds on the black-box complexity of a class of gen-

eralized OneMax functions [51, 97]. We are only considering one specific function and the black-box complexity of a single function is trivially 1 [51].

The upper bound for OneMax is asymptotically tight. In fact, the following result holds for arbitrary functions that only have one global optimum. This presents a limit to the potential speedup achievable by crossover.

> **Theorem 3.3.** *The expected optimization time of the shuffle GA on any function with a unique optimum is* $\Omega(\sqrt{n})$.

Proof. Without loss of generality, we suppose the optimum to be the all-1 string. We use drift on the number of 0-bits in the individual. For all t, we let X_t be the random variable of this number after t iterations. It is easy to see that we can bound the drift of $(X_t)_{t\in\mathbb{N}}$ from above by supposing there is no negative drift, and that each search point with less 0-bits is accepted.

The number of bits differing in both parents when performing a uniform crossover is trivially bounded by n. By Lemma 3.1 this implies that the expected decrease of the number of zeros is always bounded by $\mathrm{E}[X_t - X_{t+1} \mid X_t] = O(\sqrt{n})$.

The lower bound now follows from well known additive drift results (Lemma 2 in [71]), saying that if always $\mathrm{E}[X_t - X_{t+1} \mid X_t] \leq \delta$ holds for some $\delta > 0$ then the expected time until $X_t = 0$ is at most $\mathrm{E}[X_0]/\delta$. Here the expected number of zeros in the initial search point is $\mathrm{E}[X_0] = n/2$. Setting $\delta := O(\sqrt{n})$, we get a lower bound of $n/(2\delta) = \Omega(\sqrt{n})$. □

For $\textsc{Jump}_k$ shuffling a search point on the plateau creates another search point on the plateau. The resulting diversity is sufficient to show the following result.

> **Theorem 3.4.** *Unless initialized with* 0^n*, the expected optimization time of the shuffle GA on* $\textsc{Jump}_k$ *with* $k \leq \sqrt{n/2}$ *is* $\Theta(\sqrt{n} + 4^k)$.

This is a significant speedup, compared to the standard (1+1) EA that needs expected time $\Theta(n \log n + n^k)$ [86]. For $k := \log n$, say, this is a difference between polynomial and superpolynomial running times.

Proof of Theorem 3.4. Following the analysis of the proof of Theorem 3.2, the individual will have reached the plateau of fitness k after $O(\sqrt{n})$ generations. The probability that a single application of the shuffle crossover now produces

the optimum is the probability of choosing disjoint sets of bit positions in the two random permutations of the individual for the 0s, and then choosing a 1 at all these positions in the uniform crossover. As there are $2k$ positions where the selected parents differ, the probability for uniform crossover setting all bits to 1 is $2^{-2k} = 4^{-k}$. The probability of obtaining the right parents can be bounded as follows. Fixing the k positions for 0-bits in the first individual, consider all k positions for the second individual to be assigned sequentially. All zeros of the second individual must not be in at most $2k$ positions that are either fixed or already used. Thus, we get a lower bound on the probability of

$$\left(1 - \frac{2k}{n}\right)^k \cdot 4^{-k} \geq \left(1 - \frac{k^2}{n}\right) \cdot 4^{-k}.$$

The upper bound follows from $k^2 \leq \frac{n}{2}$.

It is easy to see that with overwhelming probability no search point with more than $n - k$ ones and less than n ones is accepted after initialization. The lower bound then follows from the general lower bound $\Omega(\sqrt{n})$ and the fact that uniform crossover of two individuals with at least k zeros each is successful with probability at most 4^{-k}. □

We get the following corollary to the theorems above.

> **Corollary 3.5.** *Unless initialized with 0^n, the expected optimization time of the shuffle GA on* OneMax *and* $\textsc{Jump}_k$ *($k \leq \frac{\log n}{4}$) is $\Theta(\sqrt{n})$.*

3.3 Population Size vs. Crossover Probability

The consideration of the shuffle GA has shown that crossover can achieve tremendous speedups given sufficient diversity . We now turn from an "ideal" setting to a more realistic GA. Without the shuffling operator one might think that a large population is necessary to provide diversity. However, if the crossover probability is small, mutation can create diversity in subsequent steps. We make this precise for the function $\textsc{Jump}_k$ by showing that small populations, even of size as small as $\mu = 2$, suffice to optimize $\textsc{Jump}_k$ efficiently.

Jansen and Wegener [86] considered a steady-state genetic algorithm as given in Algorithm 3.2. In their by now classical analysis on the function $\textsc{Jump}_k$

they disallowed mutation to create replicates of the parent. The reason is that otherwise replicates would quickly take over the population and so diversity is lost. So, when mutation does not flip any bit, the offspring is disregarded. Note that this does not apply to mutation following crossover.

Algorithm 3.2: (μ+1) GA with avoidance of replications by mutation and with/without mutation after crossover.

```
1  Initialize population 𝒫 of size μ ∈ ℕ uniformly at random;
2  choose p_c ∈ (0, 1);
3  while true do
4      Choose q ∈ [0, 1] uniformly at random;
5      if q ≤ p_c then
6          Choose y_1, y_2 uniformly at random from 𝒫;
7          y' ← uniform crossover(y_1, y_2);
8          if use mutation after crossover then
9              Flip each bit in y' independently with probability
               1/n;
10     else
11         Choose y uniformly at random from 𝒫;
12         Create y' by flipping each bit in y independently with
           probability 1/n;
13         if y = y' then
14             Continue at line 3 to avoid replicates;
15     Choose z ∈ 𝒫 with minimal fitness in 𝒫 u. a. r.;
16     if f(y') ≥ f(z) then
17         𝒫 ← 𝒫 \ {z} ∪ {y'};
```

Their result for $\textsc{Jump}_k$ reads as follows (they also prove a tail bound, but we only review the result on the expected optimization time).

> **Theorem 3.6** (Jansen and Wegener [86])**.** *Let $k = O(\log n)$. Consider the (μ+1) GA with crossover probability $p_c \leq 1/(Ckn)$ (where C is a large enough constant), and population size μ where $\mu \geq k \log^2 n$ and $\mu = n^{O(1)}$ on $\textsc{Jump}_k$. The expected optimization time is $O(\mu n(k^2 + \log(\mu n)) + 4^k/p_c)$.*

The behavior of the (μ+1) GA on the function $\textsc{Jump}_k$ is of particular interest for us, as $\textsc{Jump}_k$ cannot be optimized efficiently by mutation only, while a crossover has to rely on a sufficient diversity in the population: the population easily reaches the locally optimal plateau of fitness n, but will then need to create what we call *complementary pairs*—pairs of individuals which do not

have a 0 at a common bit position, and which can thus be recombined to reach the optimum.

In this section we give conditions on the parameters of the (μ+1) GA which lead to efficient optimization of $\textsc{Jump}_k$, as well as conditions which lead to expected superpolynomial optimization times.

First, we prove that the (μ+1) GA also works with very small populations and not too big crossover probability. In particular, the GA is effective using the smallest possible population size: $\mu = 2$. In addition, larger crossover probabilities can be used, compared to the result by Jansen and Wegener [86]. Instead of requiring $p_c \leq 1/(Ckn)$, the following proof works for $p_c \leq k/n$.

> **Theorem 3.7.** *The expected optimization time of the (μ+1) GA with or without mutation after crossover, $\mu \geq 2$, $\mu \leq n^{O(1)}$, and $p_c \leq \frac{k}{n}$ on $\textsc{Jump}_k$ with $k = o(\sqrt{n})$ is $O\big(\mu n \log n + e^{6k} \cdot \mu^{k+2} \cdot n\big)$.*

Theorem 3.7 gives the smallest upper bound for $\mu = 2$. In this case for every $\varepsilon > 0$, if $k = c \log n$ for a sufficiently small constant c depending on ε, this gives an expectation of $O(n^{1+\varepsilon})$, while mutation-based EAs without crossover still need superpolynomial time $\Omega(n^{c \log n})$.

If $\mu = 2$ and $k = c \log \log n$ for a sufficiently small c, we get $O(n \log n)$, while mutation-based EAs without crossover need time $\Omega(n^{c \log \log n})$, which is still superpolynomial.

Proof of Theorem 3.7. It is easy to show that when $\mu = n^{O(1)}$ then in expected time $O(\mu n \log n)$ the whole population contains only the optimum 1^n or search points on the plateau (cf. the proof of Theorem 5 in [86]). Note that this property will be preserved forever as all other search points have worse fitness (apart from the optimum).

A population is called *perfect* if each member has $n - k$ ones and there is a complementary pair in the population. Note that uniform crossover can only create the global optimum 1^n in a perfect population.

We describe a sequence of lucky events that results in a perfect population. We call this sequence a *trial*. The length of such a sequence is bounded. A trial is called *successful* if it leads to a crossover operation on a perfect population. In a perfect population there are two parents having k zeros at different positions. This means that their Hamming distance is $2k$ and all these bits have to be set to 1 in a uniform crossover. The probability that a successful trial creates a global optimum is then at least $\binom{\mu}{2}^{-1} \cdot 2^{-2k}$ if no

mutation is used after crossover and $\binom{\binom{\mu}{2}}{1}^{-1} \cdot 2^{-2k} \left(1 - \frac{1}{n}\right)^n$ otherwise, as the latter factor describes the probability that mutation does not flip any bit.

The expected optimization time then follows from the probability of a trial being successful, the length of a trial, and the above probability of creating an optimum at the end of a successful trial.

Consider a population where all members have $n - k$ ones. For the next at most $n + n/k$ generations we define the following events. The first events concern the first n generations, or a shorter time span if a perfect population or a global optimum is found earlier. The last event is based on the following n/k generations. This means that a trial contains at most $n + n/k$ generations. We pessimistically ignore the possibility that a perfect population or a global optimum might be reached prematurely.

E1: Within n generations no crossover is performed. This event has probability at least $(1 - p_c)^n \geq (1 - k/n)^n \geq e^{-k}(1 - o(1))$.

E2: Conditional on E1, within n generations at least k mutations are performed such that exactly two bits are flipped and the offspring has $n-k$ 1-bits. Such a mutation has probability $p := k/n \cdot (n-k)/n \cdot (1 - 1/n)^{n-2}$ as exactly one 1-bit and one 0-bit have to be flipped. Assuming n is large enough such that $(n-k)/n \cdot (1 - 1/n)^{n-2} \geq 1/3$, we have $k/(3n) \leq p \leq k/n$. The probability of the described event is at least

$$\begin{aligned}\binom{n}{k} \cdot p^k \cdot (1-p)^{n-k} &\geq \frac{n^k}{k^k} \cdot \frac{k^k}{n^k} \cdot 3^{-k} \cdot (1-p)^{n-k} \\ &\geq 3^{-k} \cdot \left(1 - \frac{k}{n}\right)^{\left(\frac{n}{k}-1\right)\cdot k} \geq (3e)^{-k}.\end{aligned}$$

E3: Conditional on E1, within n generations it never happens that mutation creates an offspring with Hamming distance larger than 2 to its parent such that the offspring has $n - k$ ones. The probability for a single such mutation is at most $\binom{k}{2} \cdot 1/n^2 \leq k^2/n^2$ as two 0-bits need to be flipped. The event E3 thus has probability at least

$$\left(1 - \frac{k^2}{n^2}\right)^n \geq 1 - O\left(\frac{k^2}{n}\right) \geq 1 - o(1)$$

as $k = o(\sqrt{n})$. Note that E2 and E3 are not independent, but $\mathsf{P}(\text{E2} \wedge \text{E3}) \geq \mathsf{P}(\text{E2}) \cdot \mathsf{P}(\text{E3})$.

E4: Assume E1, E2, and E3 happen, which implies that we have at most k generations where the population changes, due to the avoidance of replications.

E4 is the event that selection in such a generation always chooses appropriate individuals in the following sense. A parent x is selected which is part of a pair x, y of individuals that have a maximal Hamming distance in the population: $x, y = \arg\max\{H(x', y') \mid x', y' \in \mathcal{P}\}$. Also the individual to be removed is selected in such a way that the above maximum pairwise Hamming distance in the population is not decreased.

The probability of always choosing a parent as described is at least $(2/\mu)^k$ as there is at least one pair of individuals defining the maximal Hamming distance. For the same reason the probability of always choosing the individual to be removed as described is at least $((\mu-1)/(\mu+1))^k \geq (1/3)^k$. Together, the probability for E4 is at least $(2/3)^k \mu^{-k}$.

E5: Conditional on E1–E4, the maximum Hamming distance of pairs in the current population never decreases.

Due to E4 in every accepted mutation a parent x is selected such that it forms a pair of maximal Hamming distance with some other member y of the population. Let these individuals have Hamming distance $2i$. We estimate the probability that the result of mutating x increases the Hamming distance to y in an accepted 2-bit mutation. This distance increases from its current value $2i$ if mutation flips a 0-bit where x and y agree as well as a 1-bit where x and y agree. There are i bits where x is 0 and y is 1 and i bits where this is the other way round. Conditional on an accepted 2-bit mutation, as all 0-bits and 1-bits are chosen with the same probability, respectively, the probability of choosing bits where x and y agree is $(k-i)/k \cdot (n-k-i)/(n-k)$. The probability of making a sequence of these mutations is thus at least

$$\begin{aligned}\prod_{i=1}^{k-1} \left(\frac{k-i}{k} \cdot \frac{n-2k}{n-k}\right) &= \frac{(k-1)!}{k^{k-1}} \cdot \left(1 - \frac{k}{n-k}\right)^{k-1} \\ &\geq \frac{(k/e)^k}{k^k} \cdot \left(1 - \frac{k^2}{n-k}\right) \\ &\geq e^{-k} \cdot (1 - o(1)).\end{aligned}$$

E6: Conditional on E1–E5, consider the situation after a perfect population has been reached. Note that from now on we work without conditioning on the events E1–E5. So, we take a fresh look and define E6 as the event that in the following n/k generations at least one crossover happens and no mutation leads to an accepted offspring. Note that a necessary event for an accepted mutation is that at least one of k 0-bits flips. The probability for the event E6 is at least

$$1 - (1 - p_c)^{n/k} \cdot \left(1 - \frac{k}{n}\right)^{k/n} = \Omega(1).$$

If E1–E6 happen, the trial is successful. The probability of E1–E6 happening is

$$\begin{aligned} & e^{-k} \cdot (3e)^{-k} \cdot (2/3)^k \mu^{-k} \cdot e^{-k} \cdot (1 - o(1))^3 \cdot \Omega(1) \\ = \Omega & \left(2^k \cdot 3^{-2k} \cdot e^{-3k} \cdot \mu^{-k}\right). \end{aligned}$$

The length of a trial is at most $n + n/k = O(n)$ by definition. Hence, starting from a population with $n-k$ ones in every individual, the expected time until the global optimum is created is

$$O\left(2^{-k} \cdot 3^{2k} \cdot e^{3k} \cdot \mu^k \cdot n \cdot \binom{\mu}{2} \cdot 2^{2k}\right) = O\left(e^{6k} \cdot \mu^{k+2} \cdot n\right).$$

This completes the proof of the theorem. □

The (μ+1) GA is effective for small populations if the crossover probability is fairly small. In this case the explorative effects of mutation are larger than the effect of crossover subsequently compressing the convex hull of the population.

We complement this finding by an opposite result for large crossover probabilities. If the crossover probability is an arbitrary constant, the population is compressed to a single point more quickly than mutation can create diversity. This holds for up to logarithmic population sizes.

> **Theorem 3.8.** *Consider the (μ+1) GA* without *mutation after crossover on* $\textsc{Jump}_k$ *with* $k = \log n$. *For every crossover rate* $p_c = \Omega(1)$ *there is an absolute, positive constant* $C := C(p_c)$, *depending on* p_c, *such that if* $\mu \leq C \log n$ *then the expected optimization time of the (μ+1) GA is superpolynomial.*

Proof. It is easy to see that after the first $O(\mu n \log n) = o(n^2)$ generations of the GA, with high probability, the optimum is not created (since neither mutation nor crossover can bridge the gap of the $\textsc{Jump}_k$ function in this time with sufficient probability), and the whole population is on the plateau.

Suppose now we have the whole population on the plateau. We show that, with high probability, an arbitrary population collapses to copies of a single individual within a short number of steps. More precisely, for every population on the plateau with probability at least $1/\sqrt{n}$ the whole population will consist of copies of a single individual after μ generations. A sufficient condition for this to happen is that in each generation two equal parents are selected, crossover is performed, and a proper individual is chosen for removal. If there

are i equal individuals in the population, increasing this number to $i+1$ has probability at least

$$\frac{i^2}{\mu^2} \cdot p_c \cdot \frac{\mu - i}{\mu + 1}.$$

The probability of this always happening is thus at least

$$\begin{aligned}\prod_{i=1}^{\mu-1} \left(\frac{i^2}{\mu^2} \cdot p_c \cdot \frac{\mu - i}{\mu + 1} \right) &= p_c^{\mu-1} \cdot \left(\frac{1}{\mu^2(\mu+1)} \right)^{\mu-1} \cdot ((\mu-1)!)^3 \\ &\geq p_c^{\mu-1} \cdot e^{-3(\mu-1)} \left(\frac{(\mu-1)^3}{\mu^2(\mu+1)} \right)^{\mu-1}\end{aligned}$$

and as the bracketed term is $\Omega(1)$ this probability is at least $1/\sqrt{n}$ if the constant c from the preconditions is sufficiently small. The probability that this collapse never happens in $t := \sqrt{n}\mu k \ln n$ generations is at most $(1 - 1/\sqrt{n})^{\sqrt{n}k \ln n} \leq n^{-k}$.

Assuming the population has collapsed, we show that in t steps with high probability no complementary pair is created. Without mutation after crossover, crossover then cannot create the global optimum. Complementary pairs can only be created if, starting with a collapsed population, over time in total k 0-bits have been flipped. The expected number of flipping 0-bits in t mutations is tk/n. By Chernoff bounds, the probability that at least k bits are flipped in total in this time span is

$$\left(\frac{e^{n/t-1}}{(n/t)^{n/t}} \right)^{kt/n} \leq \left(\frac{e^{n/t}}{(n/t)^{n/t}} \right)^{kt/n} = \left(\frac{et}{n} \right)^k.$$

By definition of t and recalling $k = \log n$, this probability is superpolynomially small. Hence, with a superpolynomially small error probability the population collapses again before a complementary pair is created.

The probability of creating the global optimum by mutation is at most n^{-k} and hence superpolynomially small as well. As one of the mentioned unlikely events must occur in order to create the global optimum, this proves a superpolynomial lower bound. □

3.4 Experimental Results for Population Diversity

This section adds experimental results about the behavior of the (μ+1) GA on the plateau of $\textsc{Jump}_k$. We are particularly interested in the population diversity given by the distribution of Hamming distances between individuals. The most important case is the number of complementary pairs; the existence of many such pairs guarantees that uniform crossover is effective as 1^n can only be created if complementary parents are selected.

We set the crossover probability to $p_c = 0.5$ and allow mutations after a crossover step. We conduct experiments for various parameters n and with $k = \log n$, as well as different population sizes μ. All results are averaged over 100 runs for each experiment.

We measure the number of complementary pairs in each generation of the population on the plateau. In order to estimate the diversity at an equilibrium state, the optimum is removed of the function $\textsc{Jump}_k$ and each run is stopped after 1 million generations. Even for $\mu = 1024$ and $n = 1024$, the population gathers on the plateau mostly before generation $\approx 150\,000$ such that within the remaining $\approx 850\,000$ generations the population averages out to the equilibrium.

The analysis of the (μ+1) GA by Jansen and Wegener [86] only works under limited conditions of a large population size and very small crossover probability in order to maintain diversity. One might get the impression that a large crossover probability is harmful for the population diversity regardless of their size.

Figure 3.2 shows for $k \in \{5, 6, \ldots, 10\}$ how the ratio μ/n of populations size and $n := 2^k$ influences the number of complementary pairs in the population. Figure 3.3 shows the dependency of the percentage of complementary pairs, for different population size functions $\mu(n) \in \{2, 4, \log(n), \sqrt{n}, n\}$. For both plots the different values of μ on the x axis are linearly interpolated.

Our experiments do not confirm the mentioned impression. Having used a rather large crossover probability, we find that the larger the population, the more complementary pairs are available. Figure 3.2 shows that, for a reasonably large population size, a sufficiently large fraction of pairs are complementary pairs. However, Figure 3.3 shows that in order to have a constant fraction or a fraction converging to 1 the population size should depend on $n = 2^k$ in a linear fashion, i. e., $\mu = \Theta(n)$. For the tested population sizes $o(n)$ we do not see a constant fraction of the population being complementary

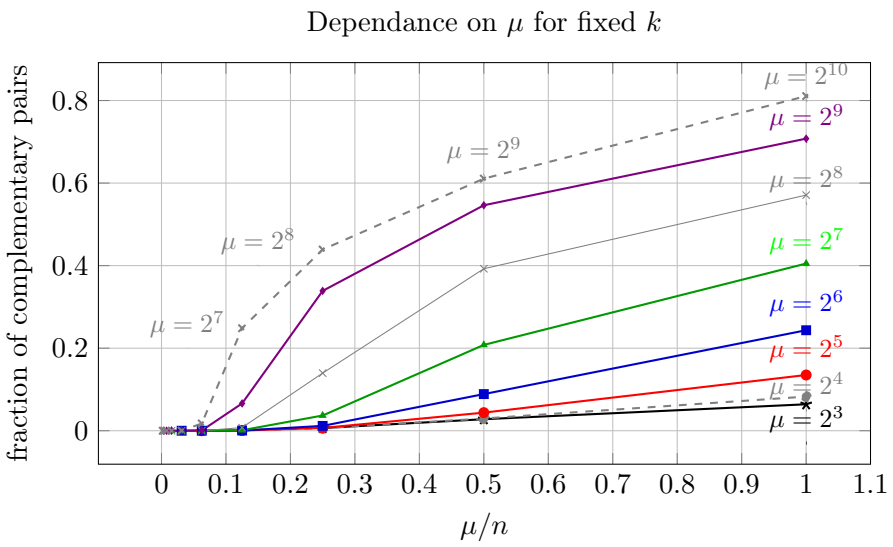


Figure 3.2: Fraction of complementary pairs vs. normalized population sizes μ/n, for the $(\mu+1)$ GA on $\textsc{Jump}_k$ with $k = 3$ (—★—), $k = 4$ (- -●- -), $k = 5$ (—●—), $k = 6$ (—■—), $k = 7$ (—▲—), $k = 8$ (—×—), $k = 9$ (—◆—), and $k = 10$ (- -★- -).

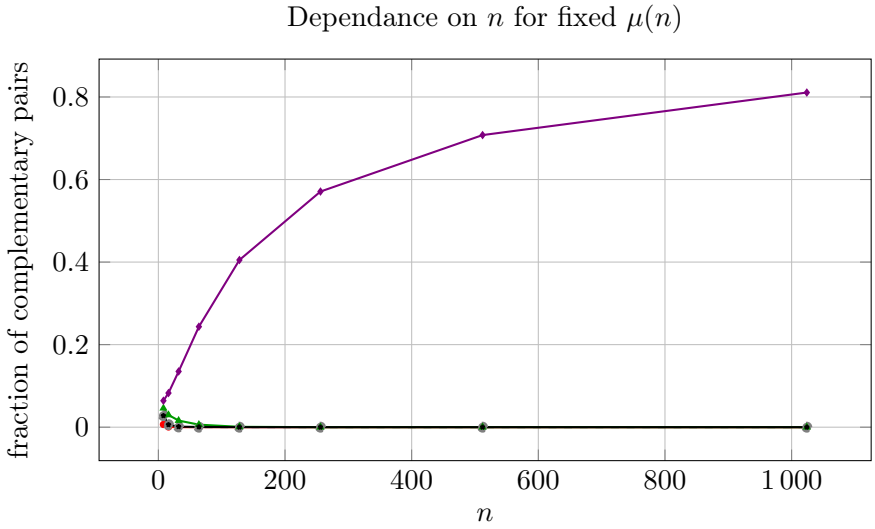
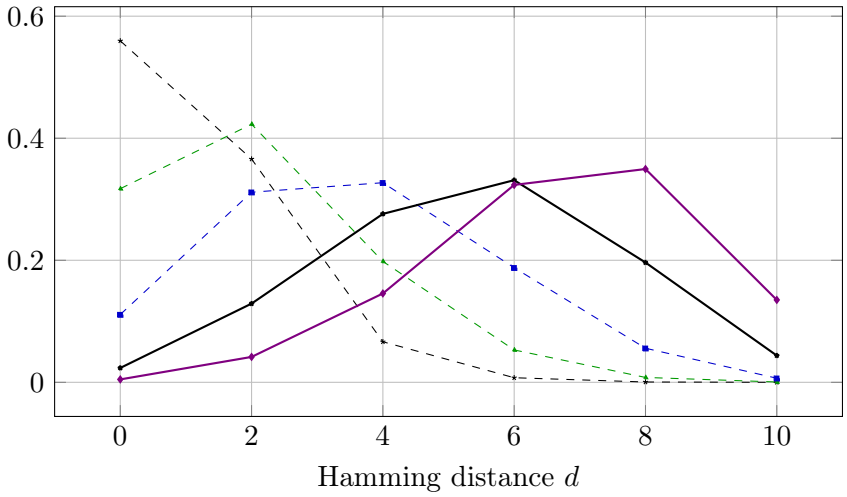


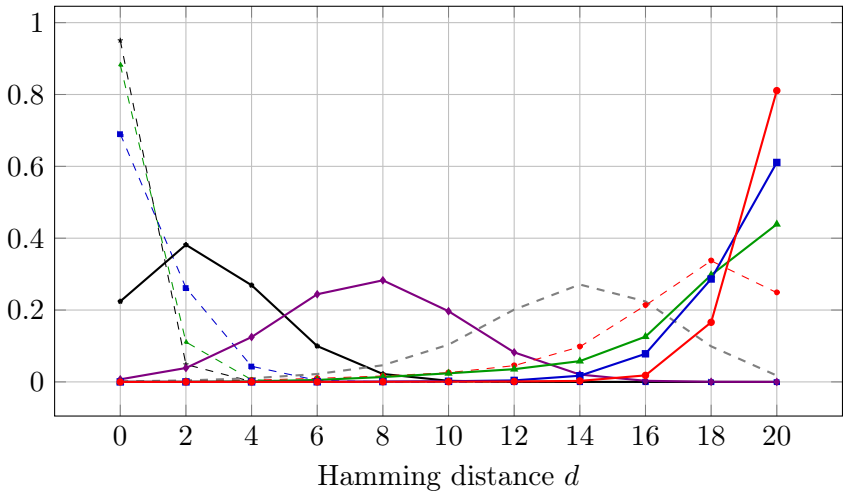
Figure 3.3: Fraction of complementary pairs vs. problem dimensions, for the $(\mu + 1)$ GA on $\textsc{Jump}_k$ with population size functions $\mu(n) = 2$ (—●—), $\mu(n) = 4$ (- -●- -), $\mu(n) = \log(n)$ (—▲—), $\mu = \sqrt{n}$ (—•—), and $\mu = n$ (—◆—).

pairs. On the other hand a population size growing linearly with n contains an overwhelming number of complementary pairs.

In order to gain further insights into the existence of complementary pairs, for $k \in \{3, \ldots, 10\}$ we computed a histogram of pairwise Hamming distances in the population, averaged over time (again with $n = 2^k$ and for vari-



(a) Hamming distance histogram for $k = 5$.



(b) Hamming distance histogram for $k = 10$.

Figure 3.4: Histograms of pairwise Hamming distances in the population, averaged over time, for the $(\mu+1)$ GA on $\textsc{Jump}_k$ with $k \in \{5, 10\}$, $n = 2^k$, and $\mu = 2$ (-·-), $\mu = 4$ (-·-), $\mu = 8$ (-·-), $\mu = 16$ (—), $\mu = 32$ (—), $\mu = 64$ (- - -), $\mu = 128$ (-·-), $\mu = 256$ (—), $\mu = 512$ (—), $\mu = 1024$ (—).

ous μ). For each generation on the plateau the pairwise Hamming distances $d \in \{0, \dots, 2k\}$ (only even d can occur) were counted and the results were averaged over all generations on the plateau. Figure 3.4 shows two selected histograms for $k = 5$ and $k = 10$; the other histograms looked similar. One can see that the curves gradually shift from leaning towards small distances to Bell curves and leaning towards large distances as the population size increases. Note that the highest value gives the average number of complementary pairs.

This also supports the claim that a large population size is needed in the presence of a large crossover probability. As can be seen here, the diversity for large population sizes is indeed significant, that is, for $k = 10$ and $\mu \geq 256$ we hardly observe any pairs of individuals having a Hamming distance less than $2k$. But we also observe the effect of a collapsing population if the population size is not large enough. This supports the claim of Theorem 3.8 even in the presence of mutation after crossover, compare the lines for $k = 10$ and $\mu \leq 64$. Here, mutation is not able to create or maintain the needed diversity opposing the effect of crossover to reduce diversity when the population size is not large enough.

With these considerations on the opposing effects of mutation and crossover within the regime not covered by our theoretical results we conclude this section and also the chapter.

4 Genetic Algorithms on Shortest Paths

The all-pairs shortest path problem (APSP) is a classical problem in computer science. Hence, [44] was a major breakthrough in the struggle of the EA theory community to design and analyze evolutionary algorithms for combinatorial optimization problems. In fact, Doerr et al. [44] for the first time came up with a genetic algorithm (with crossover) for a non-artificial combinatorial optimization problem for which they showed that a genetic algorithm is asymptotically faster than the evolutionary algorithm (with mutation only).

The work presented here not only improves [44] but also adds to the research on APSP in several other important ways. The research is joint work with several authors published to three conferences [32, 39, 111] as well as one journal [46] whose merits are discussed in Section 4.3.

After the statement of the problem in the following section we discuss in Section 4.2 previous work on shortest paths to make the difficulties in designing genetic algorithms for shortest path accessible to the reader. In Section 4.4 through 4.5.2 we introduce several algorithms highlighting different aspects of genetic algorithms and proof upper bounds on their optimization time in Section 4.6. Section 4.7 complements this line of research by an algorithm for the multi-criteria shortest paths problem together with its analysis. Additionally, in Section 4.4 we present a lower bound proof for the algorithm given there. Finally, in Section 4.8 we discuss the applicability of our results to other problems and especially the DP-problems of Chapter 2.

4.1 Statement of the Problem

Before discussing the relevant literature and our contributions we introduce the problem as well as all components needed to understand the genetic algorithm.

Let $G = (V, E)$ be a directed graph with $n := |V|$ vertices and $m := |E|$ edges. Each edge has a weight $w(e)$ assigned to it by a function $w\colon E \to \mathbb{R}$. We assume that w is conservative weight, that is, the function does not allow for negative cycles. The *all-pairs shortest path problem (APSP)* is to compute a shortest path from every vertex $u \in V$ to every other vertex $v \in V$. The problem does not allow for negative cycles in the graph because otherwise the problem becomes NP-hard (see [67]) and we cannot hope to optimize it efficiently unless P=NP. We can easily assume that the graph is complete by an easy preprocessing step which assigns each non-existent edge a weight larger than the sum of all edge weights initially present in the graph.

Algorithm 4.1: The basic Steady State GA_{APSP}

```
1  P = {P_{u,v} = (u,v) | (u,v) ∈ E};
2  while true do
3      Choose r ∈ [0,1] uniformly at random;
4      if r ≤ p_c then
5          choose two individuals P_{x,y} and P_{x',y'} from P u. a. r. ;
6          perform CROSSOVER on P_{x,y} and P_{x',y'} to obtain an
           individual P'_{s,t} ;
7      else
8          choose one individual P_{x,y} uniformly at random from P
           and perform MUTATION on P_{x,y} to obtain an individual
           P'_{s,t};
9      Perform SELECTION: if P'_{s,t} is a path from s to t then
10         if there is no individual P_{s,t} ∈ P then P = P ∪ {P'_{s,t}};
11         else if w(P'_{s,t}) ≤ w(P_{s,t}) then P = (P ∪ {P'_{s,t}}) \ {P_{s,t}};
```

A genetic algorithm uses a population of solution candidates (individuals), which are gradually improved by mutation and crossover. Here, an individual is defined as a walk, that is, a sequence of edges $(e_1, \ldots, e_k)$, where $e_i \in E$ for every $i \in \mathbb{N}$ and $k \in \mathbb{N}$. The initial population $\mathcal{I}$ consists of all paths of exactly one edge. Thus, it has a size of at most $\mu = n(n-1)$ individuals, one path for each pair of distinct vertices.

Let $\mathcal{S}$ be the *search space* of all walks up to length n. The *fitness* of an individual is a function $f\colon \mathcal{S} \to \mathbb{R}$ assigning the weight of the walk (by summing

Algorithm 4.2: The Mutation Operator for Shortest Path Problems.

Input: $P_{x,y} = (x = v_0, v_1, \dots v_{\ell-1}, y = v_\ell)$ from x to y of length ℓ

Choose a Poisson distributed random variable $s := \text{Pois}(\lambda = 1)$;

Apply the following elementary mutation $s + 1$ times;

Pick an edge $e = (u, v) \in E^-(x) \cup E^+(y) \cup \{(x, v_1), (v_{\ell-1}, y)\}$ uniformly at random;

if $e \in \{(x, v_1), (v_{\ell-1}, y)\}$ **then**

    e is removed from the individual.

else

    Otherwise for e $\in (E^-(x) \cup E^+(y)) \setminus \{(x, v_1), (v_{\ell-1}, y)$ it is appended at the corresponding end of the individual.

Output: A valid individual $P'_{v_1,y}$ or $P'_{x,v_{\ell-1}}$ of length $\ell - 1$ or a new valid individual $P'_{u,y}$ or $P'_{x,v}$ of length $\ell + 1$

up the weights of its edges) as its fitness. Individuals not representing a walk will have fitness ∞. This can happen due to the application of the crossover operator which will be defined later. The objective of the algorithm is to minimize the fitness of every possible path(u, v). Thus, a selection mechanism is required, that ensures that there is at most one path in the population for every pair (u, v) with $u \neq v$. This is achieved by removing all but the fittest individual.

The Steady State GA_{APSP} presented in Algorithm 4.1 decides in each iteration whether the offspring is produced by crossover or mutation dependent on a parameter p_c of the algorithm. With probability p_c, a crossover operator is applied to two randomly chosen individuals of $\mathcal{P}$ or otherwise (with probability $1 - p_c$) mutation is used to produce the offspring. To make sure that both operators, mutation and crossover, are used we require $p_c \notin \{0, 1\}$. For all investigations here, we assume that p_c is chosen as an arbitrary constant with $p_c \in]0, 1[$.

A new individual is created by the application of a *variation operator*, which can be mutation or crossover. *Mutation* changes an individual slightly at some random positions. The classical case is the flip of a bit within a bitstring of length n with uniform probability $\frac{1}{n}$. Such a behavior has to be simulated for more complex representations. Scharnow, Tinnefeld and Wegener [137, 138] propose Algorithm 4.2, where we denote by $E^-(v)$ and $E^+(v)$ the set of incoming and outgoing edges of a vertex v in G. In a local operation, the current path is either lengthened or shortened by a single edge leading to a new valid solution which implies that the mutation operator only constructs solutions which are paths. In this way, it is possible to simulate the binomial distribution of n trials each having probability $\frac{1}{n}$.

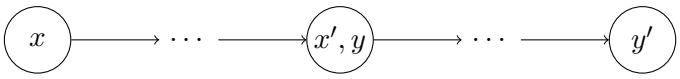


Figure 4.1: Crossover applied to two paths $P_{x,y}$ and $P_{x',y'}$ for the case $x' = y$.

The *selection operator* only accepts individuals that are paths in the graph. In addition, it ensures diversity with respect to the different pairs of vertices. For this reason, each individual $P_{u,v}$ is indexed by the start vertex u and the end vertex v. In the selection step an offspring is only compared to an individual of the current population that has the same start and end vertex. It is ensured that, for each pair of vertices (u, v) with $u \neq v$, at most one individual $P_{u,v}$ is contained in the population. This implies that the population size of our algorithms is always at most $n\,(n-1)$.

Finally, for shortest paths we define a *recombination operator* which combines at least two parents into a new individual by merging paths. Our most simple *crossover operator* for shortest paths selects two individuals i. e. paths $P_{x,x'}$ and $P_{y,y'}$ uniformly at random from the population and merges them at their common vertex into the path $P_{x,y'}$ if end vertex x' and start vertex y match, i. e. $x' = y$ holds. This operator has been introduced in [44] and reused in [32]. The situation is depicted in Figure 4.1. For the remaining case $x' \neq y$ the operator returns a dummy individual with fitness worse than all other individuals, i. e. having fitness worse than $\sum_{e \in E} w(e)$. We will discuss a tight bound for this operator in Section 4.4. In Section 4.5 we discuss more advanced types of recombination.

Having specified all components of our evolutionary cycle we come up with Algorithm 4.1 which is the prototypical algorithm for solving shortest path problems. This algorithm will be reexamined in for the different types of crossover operators and diversity mechanisms in Section 4.5, and 4.7.

4.2 Previous Work on Shortest Paths

A series of papers [32, 34, 37, 39, 42, 44, 46, 138] led to a deep understanding how evolutionary and genetic algorithms deal with path finding problems like the single source shortest path problem (SSSP) and the all-pairs shortest path problem (APSP). We will discuss previous work in this section and afterwards continue with our own contributions. The discussion of the work on shortest

paths highlights the difficulties that needed to be overcome before we were able to analyze a genetic algorithm for APSP with tight bounds.

Single Source Shortest Path Problem. Scharnow et al. [138] deal with the problem to compute for a given directed graph $G = (V, E)$ with positive edge weights a shortest path from a fixed start vertex s to all other vertices. This is the well-known Single Source Shortest Path Problem (SSSP), which can be solved by the classical Dijkstra's algorithm in time $O(n \log(n) + m)$ using Fibonacchi heaps (cf. [101]) with n the number of vertices and m the number of edges in the graph.
For the design of an EA for such a combinatorial optimization problem there are several (natural) choices for the representation of individuals, the design of a fitness function and variation operators. Scharnow et al. [138] decide for the representation $(v_1, v_2, \ldots, v_{n-1}) \in \{1, \ldots, n\}^{n-1}$, in which at position i we find the predecessor of vertex i on a shortest path from s to i. Feasible individuals thus represent a tree rooted at s. The multi-criteria fitness function consists of $n-1$ components where at position i the weight of the currently known shortest path from vertex s to vertex i is stored. With a simple (1+1) EA an optimal shortest path tree is computed within the optimization time of $O(n^3)$. Other choices for the fitness function are possible. But it is easy to see that it is not reasonable to e. g. sum up the weights of all currently known shortest paths as this would create a *needle-in-the-haystack* landscape for certain instances neglecting proper guidance to the evolutionary algorithm. For example if the edge weights are set to ∞ for all but the edges between the pairs $(i, i-1)$ which are set to 1.

The work of Doerr et al. [34, 42] improves the analysis of [138] by avoiding their level argument, additionally providing lower bounds. The authors develop an analysis method that is henceforth used in their work [44] proving crossover to be useful for shortest path problems for the first time. We believe that the intuition gained was necessary to come up with a tight bound for the all pairs shortest path problem (APSP) in [44]. The technique lives on in several other papers by different authors, e. g. in [32], and [111], as well as [46]. Note, that we make use of this strong concentration behavior[1] instead of a coupon-collector like behavior also in our proofs in Section 4.6 .

We explain this technique in more detail in the next paragraph which is the predecessor of our own work [32].

For the sake of completeness let us point out that it is also possible to design a single-criteria fitness function for the SSSP analyzed in [7]. This problem has

[1] The actual behavior of a random variable deviates only little from its expected behavior.

been stated as an open issue in [138]. The fitness function shall be computed as the sum of all shortest paths currently known combined with a not-too-large penalty term for each vertex that is not yet connected to the source node. The analysis is remarkable in the respect that it omits any argument known from Dijkstra's algorithm that is rediscovered in the behavior of the (1+1) EA for the multi-criteria fitness function. The authors argue that in each iteration the objective value decreases by a multiplicative constant leading to an upper bound on the expected optimization time of $O(n^3(\log(n) + \log(w_{\max}))$ (where $w_{\max}$ denotes an edge of maximum weight) with a almost suitable lower bound $\Omega(n^3)$.

Doerr and Johannsen [30] discuss vertex- versus edge-based representations for the SSSP. In the work better upper bounds are proved for edge-based representations of the SSSP. In a vertex-based representation edges are considered with a probability that is inverse to the indegree of their adjacent vertex which the authors claim is undesirable. A uniform distribution over all edges is achieved by designing an edge-based representation which reduces previously known optimization times for sparse graphs by a factor of $\Theta(n^2/m)$.

All-Pairs Shortest Path Problem. The same argument as in [34, 42] avoids the logarithmic factor in the mutation based part of the analysis for APSP. The key idea (in the APSP setting) is to analyze how a particular shortest path can be constructed. Such a sequence of sub-paths of the shortest path was called a trail. Again, the expected time to generate a length $\ell + 1$ path out of a length ℓ path is $O(n^3)$. By using strong concentration on the time needed to do the several small progresses encoded in the trail, it was shown that with high probability, a length ℓ path emerges from a length 1 path in time $O(\ell n^3)$ (note the missing logarithmic term compared to the usual coupon collector argument). In fact, the concentration behavior is strong enough to show that all length ℓ shortest paths are found in time $O(\ell n^3)$ with high probability (again no additional logarithmic term).

Let us now recall the results of Doerr et al. [44] and their analysis in more detail. To ease the language, let us assume in this section that for each pair of vertices, there is a unique shortest path connecting them, and let us call the number of its edges its *length*. In [44], for all $\ell \geq c \log n$, c a sufficiently large constant, it was shown that after $O(\ell n^3)$ iterations, with high probability all shortest paths of length up to ℓ are present in the population. To achieve this, only mutation is needed, though any constant crossover rate does not interfere.

It was further shown that if all shortest paths of length at most ℓ are present in the population, then with high probability after $O(\ell^{-1} n^4 \log n)$ iterations,

all shortest paths of length up to 1.5ℓ are in the population. Here, only crossover operations are needed, though again any constant mutation rate is not harmful. The logarithmic factor stems from a coupon collector type behavior. To construct a fixed shortest path of length up to 1.5ℓ, an expected number of $O(\ell^{-1}n^4)$ iterations suffice.

Now a simple calculation shows that the best optimization time is achieved by using the mutation only result up to $\ell = \Theta(\sqrt{n \log n})$ and then using the crossover only result up to $\ell = n-1$. With this setting, we have $O(n^{3.5}\sqrt{\log n})$ iterations that make use of the mutation operator and another $O(n^{3.5}\sqrt{\log n})$ iterations where only crossover is needed. The analysis indicates that there are two regimes, the first one only requiring mutation, the second one only requiring crossover and both having a constant factor overlap only.

The main question coming up when regarding the analysis described above is whether the coupon collector type argument in the crossover phase can be avoided. Note that we do not have such an argument in the mutation based part. There, one could also show that if all paths of length ℓ are present in the population, then after $O(n^3 \log n)$ rounds all paths of length $\ell + 1$ exist. Hence $O(\ell n^3 \log n)$ rounds suffice to find all paths of length up to ℓ. This type of argument is used in earlier works on evolutionary computation for path problems, e.g., the pioneering work of Scharnow, Tinnefeld and Wegener [137, 138] on the single source shortest paths (SSSP) problem.

4.3 Contribution

The question arising from the discussion in the previous section is whether a similar argument can be applied also in the analysis of crossover. For two reasons, this is non-trivial. First, a successful application of crossover needs both parents to be present in the population. Hence instead of a sequence of sub-paths (trail), we would need a binary tree to describe a possible way to construct a shortest path.

A second problem is that we have only few iterations available and cannot allow for a coupon collector type behavior. With a phase in which all shortest paths of length up to 1.5ℓ are constructed out of all shortest paths of length up to ℓ in time $O(\ell^{-1}n^4 \log n)$, the total time needed for the whole crossover phase (constructing all shortest paths out of all shortest paths of length up to $\ell = \Theta(\sqrt{n \log n})$) is of the same order of magnitude as the time needed for

the phase constructing all paths of length up to 1.5ℓ out of all paths of length up to $\ell = \Theta(\sqrt{n \log n})$.

Our contribution to this line of research is an understanding of the *interplay* between crossover and mutation, that is the simultaneous use of mutation and crossover in the analysis. The authors in [37] can only prove a weak upper bound by an analysis of phases in which either mutation or crossover is prevalent in the analysis. We show that the growth of the paths through crossover operations can be analyzed without the previously needed coupon collector behavior. Crossover constructs a sufficiently dense subset of all shortest path of a certain length, and mutation fills the remaining gaps.

Simple Crossover. To the best of our knowledge, this has never been accomplished before. Thus, we can come up with an improved upper bound for the GA. Surprisingly, this time bound is asymptotically optimal. We give a series of n-vertex graphs such that with high probability, $Cn^{3.25}(\log n)^{1/4}$ iterations do not suffice to find all shortest paths (here $C > 0$ is an absolute constant). This was the first time that a reasonable non-trivial lower bound for a genetic algorithm for a real-world combinatorial optimization problem could be shown enabling us to prove a tight bound on the algorithm.

This not only yields an improved total optimization time of $O(n^{3.25}(\log n)^{1/4})$, but also gives more insight in how crossover can be applied successfully. Given the few successful and analyzable attempts to use crossover, this is an important result. The previous analysis of Doerr et al. [44] already showed that crossover is more useful to construct longer paths than shorter ones. Here, in addition, we see that crossover has difficulties to construct all of a certain set of solutions, while it is highly efficient to construct a large and sufficiently dense subset of solutions. This set is dense enough such that mutation can fill the gaps easily. This view on crossover as being good in quickly, but sketchily exploring the search space, and on mutation as the one to fill the details might be useful for further successful applications of crossover in genetic algorithmics. Recall that APSP still is the only non-artificial problem for which a genetic algorithm using crossover successfully could be analyzed. In simple words our analysis shows that not only crossover is relatively weak in constructing short paths, it also is surprisingly weak in the construction of all paths of certain lengths, even for larger path lengths. However, this can be made up by simultaneously using mutation. Then crossover does a good job in quickly finding a dense set of long paths, and mutation fills the remaining gaps.

Repair Mechanism. In contrast to this simple algorithm, evolutionary algorithms used in practice usually employ *refined recombination strategies*. Whenever a crossover operator produces an infeasible solution, one option is to discard it. However, this typically does not lead to efficient methods, as time is wasted on producing infeasible solutions and evaluating them. The concept to avoid the creation of infeasible offspring is inspired by nature where repair of genetic material takes place all the time and is crucial for survival. The design of crossover operators, producing from two feasible solutions a new feasible one, is usually more complicated (see e. g. [102] for different crossover operators for the traveling salesman problem). To deal with this situation, one can use repair mechanisms, which produce from an infeasible solution a feasible one based on properties of both parents [159]. Another way of dealing with the problem of infeasible solutions is to use specific selection methods and/or more problem specific crossover operators that are likely to produce promising solutions [21, 103].

Typically, crossover takes two individuals and combines them into a valid path, i. e., crossover concatenates the two paths, if the end vertex of $P_{x,y}$ and the start vertex of $P_{x',y'}$ match. Choosing both individuals uniformly at random from $\mathcal{P}$, as it was done in [32, 37], often does not lead to a recombined offspring that represents a path in the given graph. In the next section, we discuss how repair mechanisms can lead to more efficient evolutionary algorithms. Later on, in Section 4.5.2, we discuss how selection methods that select promising pairs of individuals for crossover let us prove an even better upper bounds on the expected optimization time.

We study extensions of the basic algorithm by two such central concepts in recombination, namely *repair mechanisms* and *parent selection*. We show that repairing infeasible offspring leads to an improved expected optimization time of $O(n^{3.2}(\log n)^{1/5})$. As a second part of our study we prove that choosing parents that guarantee feasible offspring results in an even better optimization time of $O(n^3 \log n)$.

Both results show that already simple recombination strategies can asymptotically improve the runtime of evolutionary algorithms. We point out the effect of repair mechanisms and parent selection for crossover on the runtime of evolutionary algorithms in combinatorial optimization. We take APSP as a prominent example to show in a rigorous way how different crossover operators influence the runtime of evolutionary algorithms.

Multi-criteria Case. The multi-criteria single-source shortest path problem can be tackled by a mutation-only approach as has been shown in [75]. Horoba [75] uses the approach to subdivide the objective space into hyperboxes (cf. Definition 4.33) inspired by [151, 152].

Surprisingly, crossover is also a suitable concept for the multi-criteria[2] shortest path problem. This is the first time that rigorous runtime analyses have shown the usefulness of crossover for an *NP-hard* multi-criteria optimization problem. This is a very important contribution to the research on genetic algorithms.

We set up a framework in which we can show that our Algorithm 4.5 is an FPRAS (cf. Definition 4.32) in which no objective is worse than $(1+\varepsilon)$-times the optimum. In order to achieve this we generalize the model for APSP given in [37] to the multi-criteria APSP and analyze the corresponding multi-objective evolutionary algorithm with respect to the runtime behavior. Our results are runtime bounds that generalize the ones given in [32, 39, 46] to the multi-objective case.

Unfortunately, we cannot show that our upper bounds are tight. But nevertheless we improve the guarantee given in Horoba [75] and Neumann and Theile [111]. We omit a lower bound as general lower bounds are notoriously hard to prove. Although algorithmic lower bounds are somewhat easier they are still rare to come up with, compare the exception in Section 4.4.

We consider all three crossover operators known so far but only give a full proof for the feasible parent selection. Note, that this is an improved and so far unpublished analysis of the multi-objective shortest path problem. Finally, we link the approximation factor due to the loss of precision in the hyperboxes to the number of stages in the analysis of the algorithm. This in turn influences the population size and as a consequence also the optimization time which we discuss on page 108 for all algorithms.

And finally, in the discussion of this chapter we extend our analysis to another problem with a linear weight function. We furthermore discuss the relation of the solution method to dynamic programming problems in general.

[2] We use the terms multi-criteria and multi-objective interchangeably.

4.4 A Tight Bound on the Standard GA

In the last section we introduced a precise statement of the problem and a generic view on the genetic algorithm.

In this section we state the upper bound on the optimization time. We will defer the proof of this theorem to Section 4.6.3 where we will derive the result from the proofs of the more advanced algorithms in Section 4.5.1 and 4.5.2.

> **Theorem 4.1.** *The $(\mu + 1)$ GA using mutation and crossover with any constant crossover rate $0 < p_c < 1$ has an optimization time of $O(n^{3.25} \cdot (\log(n))^{1/4})$ with high probability.*

4.4.1 Lower Bound for the Standard GA

In this section, which appeared in [32], we show that our upper bound for the optimization time is asymptotically optimal, including the $\log^{1/4}(n)$ factor. More precisely, we present a series of graphs on n vertices such that the GA with high probability does not find the optimal solution within $Cn^{3.25}\log^{1/4}(n)$ iterations, where C is an absolute constant.

> **Theorem 4.2.** *There is an absolute constant $C > 0$ and a sequence G_n of graphs on n vertices such that for all $n \in \mathbb{N}$, the GA with high probability does not find all shortest paths in G_n in $Cn^{3.25}\log^{1/4}(n)$ iterations.*

Note that to prove the theorem, it suffices to give graphs G_n having $n(1+o(1))$ vertices. This is due to the fact that we may always add vertices connected to all other vertices via expensive edges. This does not change the set of shortest paths between the original vertices, and hence does not reduce the optimization time.

The graph series G_n that serves to prove this lower bound consists of complete bi-directed graphs with all edge weights equal to 1 or n. Let $L = \lfloor((n-1)\log(n))^{1/4}\rfloor$ and $N := \lfloor(n-1)/L\rfloor$. The graph, which is shown in Figure 4.2, contains a central vertex r (the *root*), from which N otherwise disjoint bi-directed paths emerge, which consist of L weight one edges. All edges not shown have weight n.

More formally, $G := G_n = (V, E)$ with $V = ([N] \times [L])\dot\cup\{r\}$, where $[k] := \{1, 2, \ldots, k\}$ for all positive integers k, and $E := \{(u, v) \mid u, v \in V, u \neq v\}$.

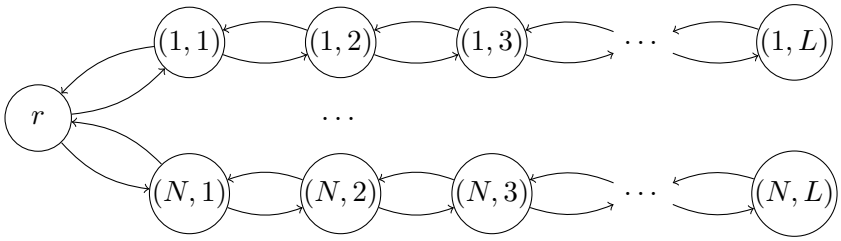


Figure 4.2: The complete bi-directed graph G. The edges shown in the figure have length 1 and the ones not depicted are longer than the shortest paths shown.

For all $i \in [N]$ and $j \in [L-1]$, the edges $(r,(i,1))$, $((i,1),r)$, $((i,j),(i,j+1))$ and $((i,j+1),(i,j))$ have weight one. All other edges have weight n. As is easy to see, G contains N shortest paths of type $r,(i,1),(i,2),\ldots,(i,L)$ and $(i,L),\ldots,(i,2),(i,1),r$, which we shall call *legs*. The set of all shortest paths between nodes x and y, that do not lie within the same leg, consists of the sub-paths between x and root r in one leg concatenated with the sub-path between r and y in the other leg. In particular, no edge with weight n lies on any shortest path. For the lower bound it is sufficient to compute the optimization time until all legs are in the population. For the N subgraphs of G, namely the legs, it was proven in [44] that the GA with mutation only has an expected optimization time of $\Omega(n^4)$, which we will cite in the following theorem.

> **Theorem 4.3** (Doerr et al. [44])**.** *The optimization time of the* $(\mu+1)$ *EA on the bidirectional* K_n *with edge weights* 1 *on a bidirected path of length* $n-1$ *and all other edge weights* n *is* $\Omega(n^4)$ *with high probability.*

The basic idea of the theorem is to use a K_n with edge weights all n except for those node u, v with $|v-u|=1$. In order to get a shortest path between the nodes 1 and n with distance $n-1$ by mutation only, the algorithm needs a number of iterations that depends on the length of the longest shortest path.

Theorem 4.2 above now follows from showing that with high probability, one of the legs is not found after $Cn^{3.25}\log^{1/4}(n)$ iterations.

> **Lemma 4.4.** *Let* C *be a sufficiently small, positive constant. Let* $n \in \mathbb{N}$ *and* $G := G_n$ *the graph used as input to the GA. Consider the population computed by the GA within* $Cn^{3.25}\log^{1/4}(n)$ *iterations. Then with high probability, not all* N *legs are in the population.*

Proof. Note that $|E| = n(n-1)$, hence already the initial population has $m = n(n-1)$ elements. In consequence, at all times a particular path is selected for mutation with probability exactly $1/m$. A pair of paths is selected for crossover with probability exactly $1/(m(m-1))$. Note further that all shortest paths in G consist of at most L edges.

Using Theorem 4.3 for any of the N legs, we may chose C in a way that $Cn^3 \cdot L$ iterations with probability $1 - \exp(-\Theta(L))$ produces no shortest path of length L. In consequence, to find all shortest paths, in each of the N legs a successful crossover has to happen.

The probability that a fixed path P of length ℓ is obtained in a single crossover operation is at most $\ell n^{-2}(n-1)^{-2}$, since there are exactly ℓ pairs of paths that can be concatenated to get P. There are $L - \ell$ paths of length ℓ per leg (which has length L), and there are N legs altogether.

Hence the probability that a single invocation of the crossover operator finds any shortest path contained in a leg, is at most $NL^3n^{-4}(1-o(1))$. Let us call this event a success in the corresponding leg. Viewing the N legs as coupons, we do not obtain all coupons in $m := (1/4)N\log(N)(NL^3n^{-4}(1-o(1)))^{-1} = (1/4-o(1))n^3 \cdot L \cdot \log(N)$. This follows from the classical occupancy problem see e. g. [58, Chapter IV.2], from which we know that the probability to get all N coupons in m iterations equals

$$\begin{aligned}
& \sum_{i=0}^{N} (-1)^i \binom{N}{i} \left(1 - \frac{i}{N}\right)^m \\
\leq\ & \sum_{i=0}^{N} (-1)^i \frac{N^i}{i!} \exp\left(\frac{-im}{N}\right) \\
\leq\ & \exp(-N) \cdot \sum_{i=0}^{N} \exp\left(\frac{-i\log(N)n^4}{NL^3(1-o(1)))}\right) \\
\leq\ & \exp(-\Omega(n^{0.75-\varepsilon})),
\end{aligned}$$

for any $\varepsilon > 0$. In consequence, with high probability $\min\{C, 1/4 - o(1)\}n^3 \cdot L$ rounds do not suffice to find all legs. □

4.5 Repair Strategies for Recombination

4.5.1 Crossover with Repair

In this section, we present a simple way to increase the success probability of the crossover operator used in previous work. This leads, as we shall prove rigorously in Section 4.6.2, to an optimization time of $O\big(n^{3.2}(\log n)^{0.2}\big)$.

The main reason why previous crossover operators for APSP have a relatively small success probability is the fact that very often the two parent individuals simply do not fit together. That is, the end-point of the first is not equal to the starting point of the second path. Since this is a rather obvious way of failing, one might think of simple solutions.

One natural way is the following. If the end-point of the first and the starting point of the second path are different, we try to bridge this gap by the (if existent, unique) path from one point to the other which is contained in our population. If the population does not contain such a bridging path, then the crossover operator still fails. This is what we shall call *crossover with repair*.

Algorithm 4.3: Crossover with Repair.

Input: $P_{x,y} = (x, \dots, y)$ and $P_{x',y'} = (x', \dots, y')$
1 **if** $y = x'$ **then**
2 $\quad P'_{s,t} \leftarrow (s = x, \dots, y = x', \dots, t = y')$ merging $P_{x,y}$ and $P_{x',y'}$ at y ;
3 **else**
4 $\quad$ **if** *there is a path* $P_{y,x'}$ *from* y *to* x' *in* $\mathcal{P}$ **then**
5 $\quad\quad P'_{s,t} = (s = x, \dots, y, \dots, x', \dots, t = y')$ merging $P_{x,y}$, $P_{y,x'}$ and $P_{x',y'}$ at their common endpoints;
6 $\quad$ **else**
7 $\quad\quad$ the operator fails and returns a dummy individual with fitness worse than all other possible individuals;

Let the crossover operator with repair be defined as follows (see Figure 4.3 for a depiction of the application of the crossover operator).

The individual $P_{y,x'}$ from Line 4 is called *repair-path*. We refer to this operator as the *crossover with repair* or repair-crossover.

Note that this operation replaces Lines 5 in Algorithm 4.1.

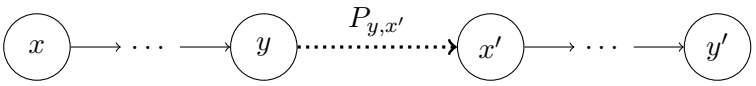


Figure 4.3: Effect of the repair crossover of Algorithm 4.3 applied to two paths $P_{x,y}$ and $P_{x',y'}$ for the case $y \neq x'$.

For the variant of the Steady State GA_{APSP} using the repair-operator, we obtain the following theorem.

> **Theorem 4.5** (Crossover with Repair)**.** *For every crossover rate $p_c \in (0,1)$ there exists a positive absolute constant $C := C(p_c)$ such that the Steady State GA_{APSP} using crossover with repair (Algorithm 4.3) has an optimization time of at most $Cn^{3.2}(\log n)^{0.2}$ w. h. p.*

A proof of this theorem can be found in Section 4.6.2. As mentioned in the introduction, this result shows that the simple repair mechanism improves on the runtime of the Steady State GA_{APSP} without repair (which had a runtime of $\Theta(n^{3.25}(\log n)^{0.25})$).

4.5.2 Feasible Parent Selection

The previous section introduced a simple repair mechanism that leads to an optimization time of $O\big(n^{3.2}(\log n)^{0.2}\big)$, which is already an improvement over the optimization time of $\Theta(n^{3.25}(\log n)^{0.25})$ for the Steady State GA_{APSP} in [32]. Nevertheless, the crossover operator may still produce solutions that do not constitute paths. This is the case if the start vertex of the second individual does not match the end vertex of the first individual and there is no individual in $\mathcal{P}$ for repair.

In the following, we want to make sure that the crossover operator constructs feasible solutions, i. e. individuals that represent paths. This is done by restricting the parent selection for crossover to individuals that match with respect to their endpoints. We choose the two individuals for crossover in Line 5 of Algorithm 4.1 using the feasible parent selection procedure given in Algorithm 4.4.

Algorithm 4.4: Feasible Parent Selection.

1 Choose $P_{x,y} \in \mathcal{P}$ uniformly at random.
2 Choose $P_{x',y'} \in \{P_{u,v} \mid P_{u,v} \in \mathcal{P} \wedge u = y \wedge v \neq x\}$ uniformly at random.

It chooses the first individual $P_{x,y}$ uniformly at random from the population $\mathcal{P}$ and the second individual $P_{x',y'}$ uniformly at random among all individuals in $\mathcal{P}$ whose start vertex equals the end vertex y of $P_{x,y}$ but whose end vertex does not equal the start vertex of $P_{x,y}$. Afterwards, in Line 6 of Algorithm 4.1, crossover is performed by concatenation (compare the result of this operation in Figure 4.1). Note that, due to the selection of the two individuals, a path from x to y' is constructed, which implies that the crossover operator only constructs feasible solutions.

This selection operator for the two parents lets us prove an even better upper bound on the optimization time than for crossover with repair as follows.

> **Theorem 4.6** (Feasible Parent Selection)**.** *For every crossover rate $p_c \in (0,1)$ there exists a positive absolute constant $C := C(p_c)$ such that the Steady State GA_{APSP} using feasible parent selection (Algorithm 4.4) has an optimization time of at most $Cn^3 \log n$ w. h. p.*

We will prove this theorem in Section 4.6.1. Recall that the upper bound given asymptotically equals the best known bound for a randomized search heuristic on APSP.

4.6 Proofs of the Runtime Bounds

In this section we introduce notations and mathematical methods which we will use for the proofs of Theorem 4.1, Theorem 4.5, and Theorem 4.6 in the following three sections. Some parts of this section have been taken from [46]. Note, that there is a full proof of its own (compare [32]) for the upper bound stated as Theorem 4.1. However, the proof in Section 4.4 has been completely rewritten to suit the methodology and notation of [46].

It turns out, that in all proofs the analysis follows a common scheme. At the center of the analysis is a stage-wise analysis of the optimization process of the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ which is inspired by the dynamic programming solutions that exist for shortest paths. That is, we show that the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ passes through certain stages during its execution until it eventually reaches a stage where the population contains only shortest paths. During these stages the paths considered are of increasing length. Moreover, they are weight-optimal with respected to that fixed length. Note that these stages are merely a concept to facilitate the runtime analysis of the Steady State

$\mathrm{GA}_{\mathrm{APSP}}$ and do not explicitly occur in the definition (Algorithm 4.1) of the Steady State $\mathrm{GA}_{\mathrm{APSP}}$.

In order to establish this result we need some tools from probability theory (cf. [2]).

Theorem 4.7 (Union Bound)**.** *Let $E_1, E_2, \ldots, E_n$ arbitrary events in some probability space. Then*

$$\mathbf{Pr}\left[\bigcup_{k=1}^{n} E_k\right] \leq \sum_{k=1}^{n} \mathbf{Pr}\left[E_k\right].$$

Theorem 4.8 (Chernoff Bound)**.** *Let $X_1, \ldots, X_t$ be mutually independent random variables with $X_i \in \{0, 1\}$ for all $i \in \{1, \ldots, t\}$. Let $X := X_1 + \cdots + X_t$. Then for all $0 < \alpha < 1$,*

$$\mathbf{Pr}\left[X < \alpha \cdot E[X]\right] < \exp\left(-\tfrac{1}{2}E[X] \cdot (1-\alpha)^2\right).$$

As in the previous sections, we again assume that we are given a a strongly connected directed graph $G = (V, E)$ on n vertices and a conservative weight function $w\colon E \mapsto \mathbb{R}$, i. e. G does not contain cycles of negative weight. Recall that the *weight* of a path is the sum of the weights of all its edges while its *length* is simply the number of its edges. To avoid confusion, we refer to (weight-) shortest paths in G as *weight-minimal*, that is, a u-v-path P is weight-minimal in G if all u-v-paths in G have at least the same weight as P.

It turns out that throughout the proofs in this and the following sections we will never regard the actual weight of a path in G (although, of course, we will be constantly concerned with the presence of certain weight-minimal paths in the population). Instead, we repeatedly make use of the following observation, which remarks that our crossover operator can exploit the optimal substructure properties of paths.

Lemma 4.9. *Let $G = (V, E)$ be a finite, strongly connected directed graph with a conservative, linear weight function $w\colon E \to \mathbb{R}$. Assume that $P_{u,x}$ and $P_{x,v}$ as well as $P'_{u,x}$ and $P'_{x,v}$ are paths in G, and $\circ$ denotes the concatenation of two paths. If $w(P'_{u,x}) \leq w(P_{u,x})$ and $w(P'_{x,v}) \leq w(P_{x,v})$ holds, then we can deduce $w(P'_{u,x} \circ P'_{x,v}) \leq w(P_{u,x} \circ P_{x,v})$.*

Proof. For our linear weight function w given by the definition of APSP we automatically get

$$\begin{aligned} w(P'_{u,x} \circ P'_{x,v}) &= w(P'_{u,x}) + w(P'_{x,v}) \\ &\leq w(P_{u,x}) + w(P_{x,v}) \\ &= w(P_{u,x} \circ P_{x,v}). \end{aligned}$$

□

Corollary 4.10. *If the concatenation of two paths $P_{u,x} \circ P_{x,v}$ creates a weight-optimal path $P_{u,v}$, and if the two paths $P'_{u,x}$ and $P'_{x,v}$ are weight-optimal, too, then $P'_{u,x} \circ P'_{x,v}$ creates a weight-optimal path $P'_{u,v}$.*

Proof. We have $w(P'_{u,x} \circ P'_{x,v}) \leq w(P_{u,x} \circ P_{x,v})$ due to Lemma 4.9 and also $w(P_{u,x} \circ P_{x,v}) \leq w(P'_{u,x} \circ P'_{x,v})$ due to the optimality of $P_{u,x} \circ P_{x,v}$. □

Note, that by design our crossover operators concatenate several paths to derive $P_{u,v}$. Consequently, there is an order in which Lemma 4.9 and Corollary 4.10 can be applied such that the property of weight-minimality holds if our crossover operators are applied to weight-minimal paths. Regarding mutation we ignore the ability to shrink paths by deleting edges at the beginning or end in our analyses. Thus, the application of mutation can also be regarded as a concatenation of two suitable paths.

In order to define the different stages properly (which we do separately in the following two sections for each of the two versions of the Steady State $\mathrm{GA}_{\mathrm{APSP}}$), we distinguish all pairs of vertices in G for which there exists a weight-minimal path of given length.

Definition 4.11 (Vertex Pairs V_a^2)**.** *For $a \in \mathbb{R}^+$, let V_a^2 be the set of all pairs $(u, v) \in V^2$ with $u \neq v$ such that among all weight-minimal u-v-paths in G there exists a path of length at most a.*

Note that in the previous definition the number a can take any (positive) *real* value although the lengths of paths in G is are always *integral*. We chose this definition for the sole reason of producing formally correct proofs which avoid tedious rounding operators. For the understanding of these proofs, however, we may think of a as being integral. In particular, we have that $V_a^2 = V_{\lfloor a \rfloor}^2$ for all $a \in \mathbb{R}^+$. Moreover, since G is strongly connected and a path in G is at most of length $n - 1$, it holds that

$$V_a^2 = \{(u, v) \in V^2 \colon u \neq v\}. \tag{4.1}$$

for all $a \in \mathbb{R}^+$ with $a \geq n - 1$.

At this point, let us also introduce a probabilistic tool we will repeatedly use in the proofs of Theorem 4.5, Theorem 4.6, Theorem 4.1, and Theorem 4.35. The following Lemma is an adaptation of the Coupon Collector argument. It allows us to give a lower bound on the probability of hitting each element of a set I using r samples, provided the probability to sample an element that has not yet been sampled is bounded below by a positive constant.

Lemma 4.12 (Coupon Collector with Dependencies)**.**
Let I be a finite set, $p \in (0,1)$, and $\{A_i^t\}_{t\in\mathbb{N}}$ be sequences of events indexed by I such that

$$\mathbf{Pr}\left[A_i^t \,\middle|\, \bigcap_{0\leq s<t} \overline{A_i^s}\right] \geq p$$

holds for all $t \in \mathbb{N}$ and all $i \in I$.

If T is the random variable that denotes the first point in time $t \in \mathbb{N}$ such that for all $i \in I$ one of the events $A_i^0, \ldots, A_i^t$ has occurred, then it holds for all $r \in \mathbb{R}^+$ that

$$\mathbf{Pr}\left[T \geq r\right] \leq |I| \cdot \mathrm{e}^{-pr}\,.$$

Proof. First, we show the lemma for $r \in \mathbb{N}$. For each $i \in I$, let B_i be the event that none of the events $A_i^0, \ldots, A_i^{r-1}$ occurs. Then

$$\mathbf{Pr}\left[B_i\right] = \mathbf{Pr}\left[\bigcap_{0\leq t<r} \overline{A_i^t}\right] = \prod_{0\leq t<r} \mathbf{Pr}\left[\overline{A_i^t} \,\middle|\, \bigcap_{0\leq s<t} \overline{A_i^s}\right]\,.$$

holds for all $i \in I$ and thus

$$\mathbf{Pr}\left[B_i\right] \leq (1-p)^r \leq \mathrm{e}^{-pr}\,.$$

Since the two events "$T \geq r$" and "$\bigcup_{i\in I} B_i$" coincide, we obtain by the Union Bound (see [2]) that

$$\mathbf{Pr}\left[T \geq r\right] \leq \sum_{i\in I} \mathbf{Pr}\left[B_i\right] \leq |I| \cdot \mathrm{e}^{-pr}\,.$$

Now, the lemma also follows for arbitrary positive real values of r,

$$\mathbf{Pr}\left[T \geq r\right] = \mathbf{Pr}\left[T \geq \lceil r \rceil\right] \leq |I| \cdot \mathrm{e}^{-p\lceil r\rceil} \leq |I| \cdot \mathrm{e}^{-pr}$$

holds for all $r \in \mathbb{R}^+$. □

In the following two sections, we present the proofs of Theorem 4.5 and Theorem 4.6, that is, we give upper bounds on the optimization times of the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ using the operator "Crossover with Repair" and the operator "Feasible Parent Selection", respectively. Since the proof of Theorem 4.5 is more involved, we start with the proof of Theorem 4.6 in the next section.

4.6.1 Proof for Feasible Parent Selection

This section is devoted to the proof Theorem 4.6. For simplicity, whenever we refer to the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ in this section, we assume without further mentioning that the operator "Feasible Parent Selection" is applied. As discussed above, we want to analyze the behavior of this algorithm in stages. To this end, we say the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ has completed the k-th stage if the population contains a weight-minimal u-v-path for every pair of vertices (u, v) for which there exists such a weight-minimal path of length at most $a(k)$ in the graph G, where the sequence $\{a(k)\}_{k\in\mathbb{N}}$ is given by

$$a(k) := \left(\frac{3}{2}\right)^k \tag{4.2}$$

for all $k \in \mathbb{N}$. Clearly, the first point in time when this event happens defines a random variable. We call this random variable T_k and say it marks the end of the k-th stage. The following definition makes this notion precise.

> **Definition 4.13** (Time T_k)**.** *For $k \in \mathbb{N}$, let $a(k)$ be as defined in* (4.2) *and let T_k be the random variable that denotes the first point in time such that, for all pairs $(u, v) \in V^2_{a(k)}$, the population of the Steady State GA_{APSP} contains a weight-minimal u-v-path.*

Observe crucially that, although $(u, v) \in V^2_{a(k)}$ implies there exists a weight-minimal path of length at most $a(k)$ in G, we only require the existence of any weight-minimal u-v-path in the population of the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ at time T_k. In particular, this u-v-path may be arbitrarily long.

It is clear by the definition of the sets $V^2_{a(k)}$ (Definition 4.11) that

$$T_0 \leq T_1 \leq T_2 \leq \dots$$

holds. Also note that, depending on G, the sets $V^2_{a(k)}$ and $V^2_{a(k+1)}$ may be equal for some values of k. In this case, also the random variables T_k and T_{k+1} coincide, that is, we have $T_k = T_{k+1}$.

We have already seen in (4.1) that if $a(k) \geq n-1$, then the population of the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ contains only weight-minimal paths. Since $a(n) \geq n-1$ for all values of n, this implies that at time T_n latest, the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ has found a population of weight-minimal paths. In other words, the random variable T_n dominates the optimization time of the Steady State $\mathrm{GA}_{\mathrm{APSP}}$. Thus, in order to show Theorem 4.6, it is sufficient to show that, for every $p_c \in (0,1)$, there exists a positive absolute constant $C := C(p_c)$ such that

$$\mathbf{Pr}\left[T_n \geq Cn^3 \log n\right] \leq \frac{1}{n}. \tag{4.3}$$

In fact, we show an even stronger statement, given in the following proposition.

Proposition 4.14. *For every $p_c \in (0,1)$, there exists an positive absolute constant $C_1 := C_1(p_c)$ such that,*

$$\mathbf{Pr}\left[T_k \geq T_{k-1} + C_1 \left(\frac{2}{3}\right)^k n^3 \log n + 1\right] \leq \frac{1}{n^2}. \tag{4.4}$$

holds for all $k \in \{1, \ldots, n\}$.

Before we prove Proposition 4.14, let us first argue how it implies (4.3) and thus Theorem 4.6. To this end, consider the telescopic sum

$$T_n = T_0 + \sum_{k=1}^{n} \left(T_k - T_{k-1}\right).$$

The random variable T_0 denotes the first point in time such that the population contains a weight-minimal path for all pairs (u, v) which have a weight-minimal path of length $a(0) = 1$. But such a path of length 1 consists only of a single (directed) edge, and for all vertex pairs that form such an edge this edge is present in the initial population (Step 1 in Algorithm 4.1). Therefore, we always have $T_0 = 0$. Next, suppose that

$$T_k - T_{k-1} \leq C_1 \left(\frac{2}{3}\right)^k n^3 \log n + 1$$

holds for all $k \in \{1, \ldots, n\}$. In this case, we have

$$T_n \leq C_1 \left(\sum_{k=1}^{n} \left(\frac{2}{3}\right)^k\right) n^3 \log n + n \leq 2C_1 n^3 \log n + n,$$

where we bounded $\sum_{k=1}^{n}(2/3)^k$ by the geometric sum

$$\sum_{k=1}^{\infty}\left(\frac{2}{3}\right)^k \leq 2.$$

Thus, if we set $C := 2C_1 + 1$, then the event

$$\text{“}\forall k \in \{1,\ldots,n\}\colon T_k - T_{k-1} \leq C_1(2/3)^k n^3 \log n + 1\text{”}$$

implies the event “$T_n \leq Cn^3 \log n$”. Conversely, this means that the event “$T_n \geq Cn^3 \log n$” implies the event

$$\text{“}\exists k \in \{1,\ldots,n\}\colon T_k - T_{k-1} \geq C_1(2/3)^k n^3 \log n + 1\text{”}.$$

We define the event E_k as $T_k - T_{k-1} \geq C_1(2/3)^k n^3 \log n + 1$, and the event “$\exists k \in \{1,\ldots,n\}\colon T_k - T_{k-1} \geq C_1(2/3)^k n^3 \log n + 1$” as $\bigcup_{k=1}^{n} E_k$ and apply the Union Bound (compare Theorem 4.7).

Therefore, we have

$$\begin{aligned}\mathbf{Pr}\left[T_n \geq Cn^3 \log n\right] &\leq \mathbf{Pr}\left[\exists k \in \{1,\ldots,n\}\colon E_k\right]\\ &\leq \mathbf{Pr}\left[\bigcup_{k=1}^{n} E_k\right]\\ &\leq \sum_{k=1}^{n}\mathbf{Pr}\left[E_k\right]\end{aligned}$$

and Inequality (4.3) follows from Proposition 4.14 by the Union Bound because we sum up at most n events with probability at most $1/n^2$.

In order to conclude the proof of Theorem 4.6, we still need to show Proposition 4.14. We devote the remainder of this section to this proof.

Proposition 4.14 basically states that with sufficiently high probability, the time needed by the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ between the $(k-1)$-th stage and the k-th stage is not too long. Interestingly, for the version of the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ we are considering in this section (the one with feasible parent selection), it is sufficient to regard only the effect of crossover and to neglect the effect of mutation (which can only be beneficial to the analysis) on the optimization process.

In our analysis, we strongly rely on the following considerations. If we perform (feasible) parent selection and subsequent crossover on a population which contains all weight-minimal paths of length at most $a(k-1)$, then it is sufficiently likely to generate each weight-minimal path of length at

most $a(k) = (3/2)a(k-1)$. Now, the crucial observation is that this argument remains valid if the population contains weight-minimal paths of *any* length for every pair of vertices that has a weight-minimal paths of length at most $a(k-1)$ in the graph G. Of course, in this case we cannot guarantee the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ produces every weight-minimal path of length at most $a(k)$. However it will instead produce some weight-minimal path for every pair of vertices that has a weight-minimal paths of length at most $a(k)$ in the graph G. This observation is formalized in the following lemma.

Lemma 4.15. *For every $a \in \mathbb{R}^+$ with $a \geq 1$ and every pair $(u,v) \in V^2_{(3/2)a} \setminus V^2_a$, there exist at least $a/4$ different vertices $x \in V$ such that (u,x) and (x,v) are in V^2_a and such that the concatenation of every weight-minimal u-x-path with every weight-minimal x-v-path at the vertex x results in a weight-minimal u-v-path.*

Proof. Let the vertex pair (u,v) be in the set $V^2_{(3/2)a}$ but not in the set V^2_a. By the definitions of V^2_a and $V^2_{(3/2)a}$, there exists a weight-minimal u-v-path $P_{u,v} = (u = u_0, u_1 \ldots, u_\ell = v)$ of length ℓ with

$$a < \ell \leq (3/2)a\,.$$

Consider the index set

$$J := \left\{\lceil a/2 \rceil, \ldots, \lfloor a \rfloor\right\}.$$

Let $j \in J$ and $x := u_j$. Note that $\lfloor a \rfloor \leq a < \ell$ and therefore u_j is a well-defined vertex of $P_{u,v}$. By Corollary 4.10, the two paths $P_{u,x} = (u = u_0, \ldots, u_j = x)$ and $P_{x,v}(x = u_j, \ldots, u_\ell = v)$ are weight-minimal, too. Moreover, both paths are of length at most a since $j \leq a$ and also $\ell - j \leq (3/2)a - a/2 = a$. Hence, (u,x) and (x,v) are in V^2_a. Furthermore, again by Lemma 4.9, the concatenation of any two weight-minimal paths $P'_{u,x}$ and $P'_{x,v}$ (of arbitrary length) at the vertex x is weight-minimal, too. Now, all that is left to do is to bound the number of possible choices for x. Since

$$|J| = \lfloor a \rfloor - \left\lceil \frac{a}{2} \right\rceil + 1 \geq \begin{cases} 1 - 1 + 1 \geq \frac{a}{4} & \text{if } 1 \leq a < 2, \\ 2 - 2 + 1 \geq \frac{a}{4} & \text{if } 2 \leq a < 4, \\ a - 1 - \frac{a}{2} - 1 + 1 \geq \frac{a}{4} & \text{if } a \geq 4, \end{cases}$$

there are at least $a/4$ ways to choose x, which concludes the proof of this lemma. □

Finally, with Lemma 4.15 at hand, we are ready to prove Proposition 4.14 which will also conclude the proof of Theorem 4.6.

Proof of Proposition 4.14. Let $k \in \{1, \dots, n\}$. We show that (4.4) holds, that is, we show that there exists a positive absolute constant $C_1 := C_1(p_c)$ (to be chosen later) such that the event

$$\text{“}\, T_k \geq T_{k-1} + 1 + C_1(2/3)^k n^3 \log n \,\text{”} \tag{4.5}$$

occurs with probability at most $1/n^2$.

At time T_{k-1}, which marks the beginning of the k-th stage, we have for every pair of vertices in $V^2_{a(k-1)}$ a weight-minimal path in the population of the Steady State $\mathrm{GA}_{\mathrm{APSP}}$. Now, we want to bound the duration of the k-th stage, that is, the number of iterations needed until the population also contains a weight-minimal path for every pair of vertices in $V^2_{a(k)}$.

As a consequence of Lemma 4.15, we will see below that in each iteration of the k-th stage the probability that the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ produces a weight-minimal path for a pair in $V^2_{a(k)} \setminus V^2_{a(k-1)}$ is at least $(1/6)(3/2)^k p_c n^{-3}$. Thus, informally speaking, we would expect a kind of Coupon Collector process to happen, which produces all such weight-minimal paths (there are at most n^2) in an expected number of $6p_c^{-1}(2/3)^k n^3 \log n^2$ iterations.

However, since we only have lower bounds on the probabilities to find a pair in $V^2_{a(k)} \setminus V^2_{a(k-1)}$ and since the events we regard are not independent of each other, we have to be more careful in bounding the probability of the event that (4.5) holds. For this reason, we apply Lemma 4.12.

In the notation of Lemma 4.12, let $I = V^2_{a(k)} \setminus V^2_{a(k-1)}$ and let $A^t_{(u,v)}$ with $t \in \mathbb{N}$ and $(u, v) \in I$ denote the event that at time $T_{k-1} + 1 + t$ there exists a weight-minimal u-v-path in the population of the Steady State $\mathrm{GA}_{\mathrm{APSP}}$. Then, we show that there exists a $p \in (0, 1)$ such that

$$\mathbf{Pr}\left[A^t_{(u,v)} \,\middle|\, \bigcap_{0 \leq s < t} \overline{A^s_{(u,v)}} \right] \geq p \tag{4.6}$$

holds for all $t \in \mathbb{N}$ and $(u, v) \in I$.

For this, first note that since (u, v) is not in $V^2_{a(k-1)}$, with positive probability there is no weight-minimal u-v-path in the population of the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ at time $T_{k-1} + t$. Therefore, the conditional probability above is well-defined.

Next, since there exists a weight minimal-path for every vertex pair in $V^2_{a(k-1)}$, Lemma 4.15 gives us that there are at least $a(k-1)/4$ distinct vertices $x \in V$ such that (u, x) and (x, v) are in $V^2_{a(k-1)}$ and the concatenation of a weight-minimal

u-x-path with a weight-minimal x-v-path yields a weight-minimal u-v-path. For each of these vertices x, the probability that the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ performs this particular concatenation at time $T_{k-1}+1+t$ is the probability to (i) perform a crossover step, (ii) choose the weight-minimal u-x-path as first parent, and (iii) then the weight-minimal x-v-path as second parent from the population at time $T_{k-1}+t$. The event (i) has probability p_c. By the definition of the operator "Feasible Parent Selection" (Algorithm 4.4) event (ii) happens with probability at least $1/n^2$ and event (iii) with probability at least $1/n$. Moreover, these bounds are independent of the event whether or not there already exists a weight-minimal u-v-path in the population at time $T_{k-1}+t$. Thus, Equation (4.6) indeed holds for all $(u,v) \in I$ and $t \in \mathbb{N}$ with

$$p := \frac{a(k-1)p_c}{4n^3} = \frac{(3/2)^k p_c}{6n^3}.$$

Finally, since the event that at time $T_{k-1}+1+t$ one of the events $A^0_{(u,v)}, \ldots, A^t_{(u,v)}$ has occurred for all $(u,v) \in I$ implies the event that $T_k \leq T_{k-1}+1+t$, we have by Lemma 4.12 with $r := C_1(2/3)^k n^3 \log n$ and $C_1 := 24/p_c$ that

$$pr = \frac{(3/2)^k p_c}{6n^3} \cdot \frac{24}{p_c} \cdot (2/3)^k n^3 \log n = 4 \log n \geq 4 \ln n$$

and therefore, since $I \leq n^2$ that

$$\mathbf{Pr}\left[T_k \geq T_{k-1} + 1 + C_1(2/3)^k n^3 \log n\right] \leq |I| \mathrm{e}^{-4 \ln n} \leq \frac{1}{n^2}.$$

This concludes the proof of Proposition 4.14 and therefore also the proof of Theorem 4.6. □

4.6.2 Proof for Crossover with Repair

We now prove Theorem 4.5. Again for the sake of brevity, whenever we refer to the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ in this section, we now assume without further mentioning that the operator "Crossover with Repair" is applied. The proof will follow the same line of argument like that in the previous section. That is, we again want to analyze the behavior of this algorithm in stages. However, this time we say the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ has completed the k-th stage if the population contains a weight-minimal u-v-path for every pair of vertices (u,v) for which there exists such a weight-minimal path of length at most $b(k)$ in

the graph G, where the sequence $\{b(k)\}_{k\in\mathbb{N}}$ is now given by

$$b(k) := 24\left(\frac{3}{2}\right)^k (n\log n)^{1/5} \tag{4.7}$$

for all $k \in \mathbb{N}$. Notice the extra $24(n\log n)^{1/5}$-factor in the definition above compared to the definition of $a(k)$ in (4.2).

Analogously to the previous section, we let the random variable T'_k mark the end of the k-th stage. Consequently, the following definition differs from Definition 4.13 only in the choice of the sequence $\{b(k)\}_{k\in\mathbb{N}}$ instead of the sequence $\{a(k)\}_{k\in\mathbb{N}}$.

> **Definition 4.16** (Time T'_k)**.** *For $k \in \mathbb{N}$, let $b(k)$ be as defined in* (4.7) *and let T'_k be the random variable that denotes the first point in time such that, for all pairs $(u, v) \in V^2_{b(k)}$, the population of the Steady State GA_{APSP} contains a weight-minimal u-v-path.*

In the previous section, we saw that the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ applying the Feasible Parent Selection operator can find all minimum-weight shortest paths necessary to complete the current stage by crossover only. For the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ which we consider in this section (the one applying the operator "Crossover with Repair"), our analysis is slightly more involved since it includes both, crossover and mutation.

To capture the interplay between crossover and mutation, we divide all stages into two phases, which we call the *crossover* phase and the *mutation phase*. Like before with the stages, the distinction of phases are merely a method of analysis and do not change the definition of Algorithm 4.1. This means that in our analysis of the optimization behavior of the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ we will only consider the effect of crossover during the crossover phases and only the effect of mutation during the mutation phases. In the actual run of the algorithm, however, both crossover and mutation are likely to happen in all phases. Still, since both operators never remove a weight-minimal path from the population, we are safe to ignore them in our analysis.

In order to define the two phases, we now define the notions of *gaps* and *approximating paths*[3], two concepts which were introduced in [32]. The key observation behind these notions is that it suffices that crossover finds a path that sufficiently well approximates a sought-after path, because mutation is fast enough to fill the gaps.

[3]This term is not to be confused with the notion of approximation guarantee in algorithm theory where it refers to the quality of algorithms.

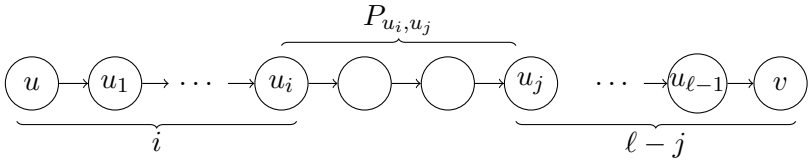


Figure 4.4: P_{u_i,u_j} is an approximating path for the vertex pair (u, v) with a gap of $g := \max\{i, \ell - j\}$.

Definition 4.17 (Approximating Path with Gap g)**.** *Let $P_{u,v} = (u = u_0, u_1, \ldots, u_\ell = v)$ be a weight-minimal path in G of length ℓ. Suppose that $P_{x,y}$ is a path in G with $x = u_i$ and $y = u_j$ for some indices $0 \leq i < j \leq \ell$. Then we call $P_{x,y}$ an* approximating *path for the pair (u, v) with (integral)* gap $g := \max\{i, \ell - j\}$*; compare Figure 4.4.*

Notice crucially that it is not necessary that an approximating path for the pair (u, v) approximates the *weight* of a weight-minimal u-v-path. In particular, every u-v-path is an approximating path with gap 0 for the vertex pair (u, v).

With the notion of approximating paths at hand, we define the two phases of the k-th stage. The first phase, which we call the *crossover phase*, starts with the beginning of the k-th stage (at time T'_{k-1}) and ends when for all pairs $(u, v) \in V^2_{b(k)}$, there exists a weight-minimal approximating path for (u, v) with gap at most

$$g(k) := \left(\frac{5}{6}\right)^k (n \log n)^{1/5} \tag{4.8}$$

for all $k \in \mathbb{N}$.

The second phase, the *mutation* phase, lasts from the end of the crossover phase to the end of the k-th stage. Corresponding to T'_k, we define the random variable T''_k which marks the end of the crossover phase and the beginning of the mutation phase.

Definition 4.18 (Time T''_k)**.** *For $k \in \mathbb{N}$ with $k \geq 1$, let $g(k)$ be as defined in* (4.8) *and let T''_k be the random variable that denotes the first point in time after T'_{k-1} such that, for all pairs $(u, v) \in V^2_{b(k)}$, the population of the Steady State GA_{APSP} contains a weight-minimal approximating path for the vertex pair (u, v) with gap at most $g(k)$.*

It is clear by Definition 4.16 and Definition 4.18 that

$$T'_0 \leq T''_1 \leq T'_1 \leq T''_2 \leq T'_2 \leq \dots \tag{4.9}$$

holds and that some of these inequalities may happen to be tight. Like in the previous section, it still holds that at time T'_n, the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ has found a population of weight-minimal paths. This time, however, we bound the number of phases more carefully. Let $k^* := k^*(n)$ be the minimum integer k such that $b(k) \geq n-1$. Then already at time T'_{k^*} the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ has found a population of weight-minimal paths. It is easy to see that

$$k^* \leq 2 \log n. \tag{4.10}$$

In order to show Theorem 4.5, it is again sufficient to show that, for every $p_c \in (0,1)$, there exists a positive absolute constant $C := C(p_c)$ such that

$$\mathbf{Pr}\left[T'_{k*} \geq Cn^3 (n \log n)^{1/5}\right] \leq \frac{1}{n}. \tag{4.11}$$

Again, we show a stronger statement. The following proposition is the direct counterpart to Proposition 4.14 in the previous section. However, this time we regard both, crossover and mutation. Both operators have constant probability to be applied in an iteration, and neither can decrease the fitness of an individual. Hence we may occasionally only regard the effect of one of the two.

We start with the consideration of the effects of mutation only. We apply a result in [44] to arrive at a population that contains with high probability a weight-minimal path for every vertex pair for which there exists a weight-minimal path in G of length at most $24(n \log n)^{1/5}$. This marks the end of the 0th-stage.

The duration of this initial stage is comparable to the duration of all next stages, during which the interplay between crossover and mutation plays a role. From this point on we consider the effects of the repair-crossover which allows for the creation of approximating weight-minimal path in the crossover phase of a stage. In the subsequent mutation phase, the mutation operator will close the gap between the approximating weight-minimal paths and the weight-minimal paths we are actually aiming for (again, we do not give a full proof for this but refer to [44] instead). The following proposition gives a precise statement of these observations for the initial mutation stage (Equation (4.12)), the k-th crossover phase (Equation(4.13)) and the k-th mutation phase (Equation (4.14)).

Proposition 4.19. *For every $p_c \in (0,1)$, there exists three positive absolute constants $C_1 := C_1(p_c)$, $C_2 := C_2(p_c)$, and $C_3 := C_3(p_c)$ such that the three inequalities*

$$\mathbf{Pr}\left[T'_0 \geq C_1 n^3 (n\log n)^{1/5} + C_1 n^3 \log n\right] \leq \frac{1}{n^2}, \tag{4.12}$$

$$\mathbf{Pr}\left[T''_k - T'_{k-1} \geq C_2 (4/5)^{2k} n^3 (n\log n)^{1/5} + 1\right] \leq \frac{1}{n^{2302}}, \tag{4.13}$$

$$\mathbf{Pr}\left[T'_k - T''_k \geq C_3 (5/6)^k n^3 (n\log n)^{1/5} + C_3 n^3 \log n\right] \leq \frac{1}{n^2} \tag{4.14}$$

hold for all $k \in \{1,\ldots,k^*\}$.

As before, we defer the proof of Proposition 4.19 until we have shown how Proposition 4.19 implies (4.11) and thus Theorem 4.5. This follows exactly the same line of proof as in the previous section. We consider the telescopic sum

$$T'_{k*} = T'_0 + \sum_{k=1}^{k^*} \left(T'_k - T''_k\right) + \sum_{k=1}^{k^*} \left(T''_k - T'_{k-1}\right).$$

and note that this time we do not necessarily have $T'_0 = 0$ due to the initial stage. Consider the event that the three inequalities

$$\begin{aligned} T'_0 &\leq C_1 n^3 (n\log n)^{1/5} + C_1 n^3 \log n, \\ T''_k - T'_{k-1} &\leq C_2 (4/5)^{2k} n^3 (n\log n)^{1/5} + 1, \\ T'_k - T''_k &\leq C_3 (5/6)^k n^3 (n\log n)^{1/5} + C_3 n^3 \log n \end{aligned}$$

hold for all $k \in \{1,\ldots,k^*\}$. If this event occurs, then we have

$$\begin{aligned} \sum_{k=1}^{k^*} (T''_k - T'_{k-1}) &\leq \Big(\sum_{k=1}^{k^*} (4/5)^{2k}\Big) C_2 n^3 (n\log n)^{1/5} + k^*, \\ \sum_{k=1}^{k^*} (T'_k - T''_k) &\leq \Big(\sum_{k=1}^{k^*} (5/6)^k\Big) C_3 n^3 (n\log n)^{1/5} + C_3 k^* n^3 \log n. \end{aligned}$$

Substituting $k^* \leq 2\log n$ and bounding the geometric series yields

$$T'_{k*} \leq (C_1 + (16/9)C_2 + 5C_3) n^3 (n\log n)^{1/5} + C_1 n^3 \log n + 2C_3 n^3 (\log n)^2 + 2\log n.$$

There exists a positive absolute constant D such that

$$C_1 n^3 \log n + 2C_3 n^3 (\log n)^2 + 2\log n \leq D n^3 (n\log n)^{1/5}$$

holds for all $n \in \mathbb{N}$. Then, for $C := 2 + C_1 + (16/9)C_2 + 5C_3 + D$, we apply the same union bound argument as in the previous section. Note that we consider the union of $2k^* + 1$, that is, of at most n events. Thus (4.11) follows from the three inequalities (4.12), (4.13), and (4.14) in Proposition 4.19. Since inequality (4.11) implies Theorem 4.5, we may conclude the proof of Theorem 4.5 by showing Proposition 4.19.

To derive the two inequalities (4.12) and (4.14) in Proposition 4.19, we apply a result from [44], which we do not prove here but simply state as Lemma 4.20. We adapt the notation to our needs, a closer look into the proofs in [44] shows that this is easily possible.

Lemma 4.20 (Analysis of Mutation)**.** *For all $p_c \in (0, 1)$ there exists a positive absolute constant $C := C(p_c)$ such that the following statement holds.*

Let $g \in \mathbb{R}^+$ and let $W \subseteq V_n^2$. Suppose that at time $t_0 \in \mathbb{N}$ the population of the Steady State GA_{APSP} contains a weight-minimal approximating path with gap at most g for every vertex pair $(u, v) \in W$. Let T be the random variable that denotes the first point in time such that the population of the Steady State GA_{APSP} contains a (proper) weight-minimal path for every vertex pair $(u, v) \in W$. Then

$$\mathbf{Pr}\left[T \geq t_0 + Cn^3(g + \log n)\right] \leq \frac{1}{n^2}.$$

The two inequalities (4.12) and (4.14) in Proposition 4.19 directly follow from the previous lemma.

Proof of Inequality (4.12) *in Proposition 4.19.* Let $C_1 := 24C$ where $C := C(p_c)$ is the positive absolute constant provided by Lemma 4.20. Then inequality (4.12) follows from Lemma 4.20 if we set $g := b(0)$, $W := V_{b(0)}^2$, and $t_0 = 0$. □

Proof of Inequality (4.14) *in Proposition 4.19.* Let $k \in \{1, \ldots, k^*\}$ and let $C_3 := C$ where $C := C(p_c)$ is the positive absolute constant provided by Lemma 4.20. Then Inequality (4.14) again follows from Lemma 4.20 if we set $g := g(k)$, $W := V_{b(k)}^2$, and replace t_0 by the random variable T_k'' (we may do this by the law of total probability, since Lemma 4.20 holds for every choice of t_0). □

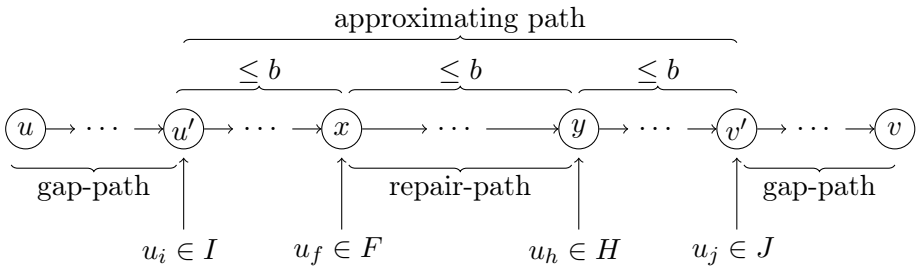


Figure 4.5: The path depicts the situation of Lemma 4.21 with two possible gap paths and a repair path from x to y. The x-y-repair-path stems from the paths from u' to x and from y to v' which are input of the repair-crossover. Each of these three paths has length at most b and their concatenation is an approximating path with gap g for the vertex pair (u, v).

At this point, we are only left to prove Inequality (4.13) in Proposition 4.19. This is done similarly to the proof of Proposition 4.14 and will be the remainder of this section. In particular, we next show a lemma which is the direct counterpart to Lemma 4.15 in the previous section.

The following lemma gives a lower bound on the number of different combinations how the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ can produce a weight-minimal approximating path with gap $g(k)$ for any weight-minimal path of length at most $b(k) = (3/2)b(k-1)$. Again, we will assume that for every weight-minimal path of length at most $b(k-1)$ there exist a weight minimal path (of arbitrary length) in the population of the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ at that time.

Lemma 4.21. *For every $b \in \mathbb{R}^+$ with $b \geq 24$, every $g \in \mathbb{R}^+$ with $g \leq b/12$, and every pair $(u, v) \in V^2_{(3/2)b} \setminus V^2_b$, there exists at least $(bg)^2 \cdot 36^{-1}$ different tuples (u', x, y, v') of four distinct vertices $u', x, y, v' \in V$ such that the pairs (u', x), (x, y), and (y, v') are in V^2_b and such that the concatenations of every weight-minimal u'-x-path and every weight-minimal y-v'-path with every weight-minimal x-y-path at the vertices x and y results in a weight-minimal u'-v'-path which is an approximating path with gap at most g for the pair (u, v); compare Figure 4.5.*

Proof. Let the vertex pair (u, v) be in the set $V^2_{(3/2)b}$ but not in the set V^2_b. By the definitions of V^2_b and $V^2_{(3/2)b}$, there exists a weight-minimal u-v-path $P_{u,v} = (u = u_0, u_1 \ldots, u_\ell = v)$ of length ℓ with

$$b < \ell \leq (3/2)b\,.$$

Consider the four index sets

$$\begin{aligned} I &:= \{0, \dots, \lfloor g \rfloor\}, \\ F &:= \{\lceil b/2 \rceil, \dots, \lceil b/2 \rceil + \lceil b/6 \rceil - 1\}, \\ H &:= \{\lceil b/2 \rceil + \lceil b/6 \rceil, \dots, \lceil b/2 \rceil + 2\lceil b/6 \rceil - 1\}, \\ J &:= \{\ell - \lfloor g \rfloor, \dots, \ell\}. \end{aligned}$$

Since $g \leq b/12$ and $b \geq 24$, we have that

$$g \leq b/12 \leq \lceil b/2 \rceil,$$

and, since $\ell > b$, we have that

$$\lceil b/2 \rceil + 2\lceil b/6 \rceil - 1 \leq 5b/6 + 2 \leq b - (b/12) < \ell - \lfloor g \rfloor, \tag{4.15}$$

the index sets I, F, H, and J are disjoint and subsets of $\{0, \dots, \ell\}$. Let $i \in I$, $f \in F$, $h \in H$, and $j \in J$. Furthermore, let $u' := u_i$, $x := u_f$, $y := u_h$, and $v' := u_j$. Then, by repeated application of Lemma 4.9 and Corollary 4.10, the three paths $P_{u',x} = (u' = u_i, \dots, u_f = x)$, $P_{x,y} = (x = u_f, \dots, u_h = y)$ and $P_{y,v'} = (y = u_h, \dots, u_j = v')$ are weight-minimal, too. Moreover, all three paths are of length at most b since, by (4.15), we have both $f \leq h \leq b$ and $h - f \leq h \leq b$. Furthermore, as in the proof of Lemma 4.15, we have also also $\ell - h \leq b$. Hence, (u', x), (x, y), and (y, v') are in V_b^2. Furthermore, again by repeated application of Lemma 4.9, the concatenation of any three weight-minimal paths $P'_{u',x}$, $P'_{x,y}$ and $P'_{y,v'}$ (of arbitrary length) at the vertices x and y is weight-minimal, too. Finally, there are

$$|I| \cdot |F| \cdot |H| \cdot |J| = (\lfloor g \rfloor + 1)^2 \lceil b/6 \rceil^2 \geq \frac{g^2 b^2}{36}$$

ways to choose the tuple (u', x, y, v'), which concludes the proof of this lemma. □

Finally, all we are left to do is to prove of Inequality (4.13) in Proposition 4.19. As announced above, this proof will again follow the lines of the proof of Proposition 4.14 in the previous section.

Proof of Inequality (4.13) *in Proposition 4.19.* Let $k \in \{1, \dots, k^*\}$. We show that (4.13) holds, that is, we show that there exists a positive absolute constant $C_2 := C_2(p_c)$ (to be chosen later) such that the event

$$\text{“ } T''_k \geq T'_{k-1} + 1 + C_2(4/5)^{2k} n^3 (n \log n)^{0.2} \text{ ”} \tag{4.16}$$

occurs with probability at most $1/n^2$.

At time T'_{k-1}, which marks the beginning of the k-th stage, we have for every pair of vertices in $V^2_{b(k-1)}$ a weight-minimal path in the population of the Steady State $\mathrm{GA}_{\mathrm{APSP}}$. Now, we want to bound the duration of the crossover phase of the k-th stage, that is, the number of iterations needed until the population also contains a weight-minimal approximating path with gap at most $g(k)$ for every pair of vertices in $V^2_{b(k)}$.

We again apply Lemma 4.12, let $I = V^2_{b(k)} \setminus V^2_{b(k-1)}$ and let $A^t_{(u,v)}$ with $t \in \mathbb{N}$ and $(u,v) \in I$ denote the event that at time $T'_{k-1}+1+t$ there exists a weight-minimal approximating path with gap at most $g(k)$ for the pair (u,v) in the population of the Steady State $\mathrm{GA}_{\mathrm{APSP}}$. We show that there exists a $p \in (0,1)$ such that

$$\mathbf{Pr}\left[A^t_{(u,v)} \,\middle|\, \bigcap_{0\le s<t} \overline{A^s_{(u,v)}}\right] \ge p \tag{4.17}$$

holds for all $t \in \mathbb{N}$ and $(u,v) \in I$.

For this, first note that since (u,v) is not in $V^2_{b(k-1)}$, with positive probability there is no weight-minimal u-v-path in the population of the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ at time $T'_{k-1}+t$. Therefore, the conditional probability above is again well-defined.

Next, since there exists a weight-minimal path for every vertex pair in $V^2_{b(k-1)}$, Lemma 4.21 gives us that there are at least $(b(k-1)g(k))^2 \cdot 36^{-1}$ distinct tuples (u',x,y,v') of vertices $u',x,y,v' \in V$ such that (u',x), (x,y), and (y,v') are in $V^2_{b(k-1)}$ and the concatenation of a weight-minimal u'-x-path, a weight-minimal x-y-path, and a weight-minimal y,v'-path at x and y yields a weight-minimal u'-v'-path which approximates the pair (u,v) with gap at most $g(k)$. Note that, in order to apply Lemma 4.21, we need here that $b(k-1) \ge 24$, which holds since

$$b(k-1) \ge b(0) = 24(n \log n)^{0.2}$$

and that $g(k) \le {}^{b(k-1)}/_{12}$, which holds since $g(k)$ is monotonically decreasing and $b(k)$ is monotonically increasing with k with $g(0) \le b(0)$.

For each of these vertex tuples (u',x,y,v'), the probability that the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ performs this particular concatenation at time $T'_{k-1}+1+t$ is the probability to (i) perform a crossover step, (ii) choose the weight-minimal u'-x-path as first parent, and (iii) then the weight-minimal y-v'-path as second parent from the population at time $T'_{k-1}+t$. The event (i) has probability p_c. By the definition of the operator "Crossover with Repair" (Algorithm 4.3) event (ii) happens with probability at least ${}^1/_{n^2}$ and event (iii) happens again with probability at least ${}^1/_{n^2}$. Moreover, these bounds are independent of the event whether or not there already exists a weight-minimal approximating

path of gap at most g for the pair (u, v) in the population at time $T'_{k-1} + t$. Thus, Equation (4.17) indeed holds for all $(u, v) \in I$ and $t \in \mathbb{N}$ with

$$p := \frac{(b(k-1)g(k))^2 p_c}{36n^4} = \frac{64(5/4)^{2k} p_c (n \log n)^{4/5}}{9n^4}.$$

Finally, since the event that at time $T'_{k-1} + 1 + t$ one of the events $A^0_{(u,v)}, \ldots, A^t_{(u,v)}$ has occurred for all $(u, v) \in I$ implies the event "$T''_k \leq T'_{k-1} + 1 + t$", we have by Lemma 4.12 with $r := C_2(4/5)^{2k} n^3 (n \log n)^{0.2}$ and $C_2 := \frac{324}{p_c}$ that

$$pr = \frac{64(5/4)^{2k} p_c (n \log n)^{4/5}}{9n^4} \cdot \frac{324}{p_c} \cdot (4/5)^{2k} n^3 (n \log n)^{0.2} = 2304 \log n \geq 2304 \ln n$$

and therefore, since $I \leq n^2$ that

$$\mathbf{Pr}\left[T''_k \geq T'_{k-1} + 1 + C_2(4/5)^{2k} n^3 (n \log n)^{1/5}\right] \leq |I| \mathrm{e}^{-2304 \ln n} \leq \frac{1}{n^{2302}}.$$

This concludes the proof of Proposition 4.19 and therefore also the proof of Theorem 4.5. □

4.6.3 Proof for the Basic Steady State $\mathrm{GA}_{\mathrm{APSP}}$

In this section we prove Theorem 4.1 along the lines of Theorem 4.5. Note, that the result of Theorem 4.1 was first stated and proved in [32] as the earliest of its kind considering the interplay between crossover and mutation. We prove the results in a reverse chronological order to increase readability. However, the importance of this result stems from the fact that we achieve a tight bound for the Steady State Steady State $\mathrm{GA}_{\mathrm{APSP}}$ applying the most basic recombination operator as defined on page 63.

Again, we will analyze the algorithm in stages. This time we say that the algorithm has completed the k-th stage if for every pair of vertices $(u, v) \in V^2$ there exists a weight-minimal path of length at most $c(k)$ in the graph G. We define the sequence $\{c(k)\}_{k \in \mathbb{N}}$ similar to Equation (4.7) as

$$c(k) := 24 \cdot \left(\frac{3}{2}\right)^k \cdot (n \log n)^{1/4}. \tag{4.18}$$

Now, we are able to define the random variable $\hat{T}'_k$ which marks the end of the k-th stage similar to Definition 4.16 with Equation (4.18) instead of (4.7).

Definition 4.22 (Time $\hat{T}'_k$)**.** *For $k \in \mathbb{N}$, let $c(k)$ be as defined in* (4.7) *and let $\hat{T}'_k$ be the random variable that denotes the first point in time such that, for all pairs $(u, v) \in V^2_{c(k)}$, the population of the Steady State GA_{APSP} contains a weight-minimal u-v-path.*

The notion of an *approximating path* from Definition 4.17 was first introduced in [32] which this section is based on. Note again that it is not necessary that an approximating path for the pair (u, v) approximates the *weight* of a weight-minimal u-v-path. It is sufficient to have a subpath of $P_{u,v}$ at hand. In particular, every u-v-path is an approximating path with gap 0 for the vertex pair (u, v).

With the notion of approximating paths the k-th stage is subdivided into two phases in which we will only regard the effects of one operator and ignore the contribution of the other. The first phase, again called the *crossover phase*, starts with the beginning of the k-th stage (at time $\hat{T}'_{k-1}$) and ends when for all pairs $(u, v) \in V^2_{c(k)}$, there exists a weight-minimal approximating path for (u, v) with gap at most

$$g(k) := \left(\frac{2}{3}\right)^{k/3} (n \log n)^{1/4} \tag{4.19}$$

for all $k \in \mathbb{N}$, where the value is taken from Doerr and Theile [32, Lemma 3].

With the crossover operator the search space is explored at large followed by a mutation phase roaming the closer neighborhood as discussed in the introduction for the working principles of crossover. The second phase, again called the *mutation* phase, lasts from the end of the crossover phase to the end of the k-th stage. We define the random variable $\hat{T}''_k$ which marks the end of the crossover phase and the beginning of the mutation phase.

Definition 4.23 (Time $\hat{T}''_k$)**.** *For $k \in \mathbb{N}$ with $k \geq 1$, let $g(k)$ be as defined in* (4.19) *and let $\hat{T}''_k$ be the random variable that denotes the first point in time after $\hat{T}'_{k-1}$ such that, for all pairs $(u, v) \in V^2_{c(k)}$, the population of the Steady State GA_{APSP} contains a weight-minimal approximating path for the vertex pair (u, v) with gap at most $g(k)$.*

The two following inequalities are comparable to Inequality (4.9) which states the connection between the random variables T'_k and T''_k. They trivially also hold for our Definitions 4.22 and 4.23.

$$\hat{T}'_0 \leq \hat{T}'_1{}' \leq \hat{T}'_1 \leq \hat{T}''_2 \leq \hat{T}'_2 \leq \dots \tag{4.20}$$

The Steady State $\mathrm{GA}_{\mathrm{APSP}}$ certainly has found a population of weight-minimal paths for $k^* := k^*(n)$ the minimum integer k such that $c(k) \geq n$. Thus, we needn't compute the optimization time for $\hat{T}'_n$ but are already finished at time $\hat{T}'_{k^*}$. Then the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ has found a population of weight-minimal paths, and it is easy to see that

$$k^* \leq \log_{1.5}(n) \leq 2 \log n. \tag{4.21}$$

Analogously to Inequality (4.11) we proof Theorem 4.1 if we are able to show that, for every $p_c \in (0,1)$, there exists an absolute positive constant $C := C_1(p_c)$ depending on the crossover rate such that

$$\mathbf{Pr}\left[\hat{T}'_{k^*} \geq Cn^3 \cdot (n \log n)^{1/4}\right] \leq \frac{1}{n}. \tag{4.22}$$

Similarly to Proposition 4.19 in Proposition 4.24 we consider an initial mutation stage ignoring the positive effects of crossover within this 0-th stage. Applying Theorem 6 of [44] respectively Lemma 4.20, at time $\hat{T}'_0$ we arrive at a population containing a weight-minimal path of length at most $24(n \log n)^{1/4}$ for every vertex pair[4]. The optimization time of this initial stage is an upper bound on the optimization times of the remaining stages which gradually get shorter. Again as in Proposition 4.19 after the initial mutation phase (Equation (4.23)) in stage k we consider the interplay between crossover and mutation within the k-th crossover phase (Equation (4.24)) as well as the k-th mutation phase (Equation (4.25)). Note that the main differences in the optimization time of the phases and stages as well as the success probability compared to Section 4.5 are due to the success probability of the basic crossover operator.

Proposition 4.24. *For every $p_c \in (0,1)$, there exists three positive absolute constants $C_1 := C_1(p_c)$, $C_2 := C_2(p_c)$, and $C_3 := C_3(p_c)$ such that the three inequalities*

$$\mathbf{Pr}\left[\hat{T}'_0 \geq C_1 n^3 \overbrace{(n \log n)^{1/4}}^{c(0)} + C_1 n^3 \log n\right] \leq \frac{1}{n^2}, \tag{4.23}$$

$$\mathbf{Pr}\left[\hat{T}''_k - \hat{T}'_{k-1} \geq C_2 (2/3)^{k-1} n^3 (n \log n)^{1/4} + 1\right] \leq \frac{1}{n^2}, \tag{4.24}$$

[4] By a preprocessing step we can assume that G is complete.

$$\mathbf{Pr}\left[\hat{T}'_k - \hat{T}''_k \geq C_3 n^3 \underbrace{(5/6)^k (n\log n)^{1/4}}_{g(k)} + C_3 n^3 \log n\right] \leq \frac{1}{n^2} \tag{4.25}$$

hold for all $k \in \{1, \ldots, k^*\}$.

Again, Proposition 4.24 implies 4.22 and hence Theorem 4.1. Let us assume the event that the following inequalities hold for all $k \in \{1, \ldots, k^*\}$

$$\begin{aligned} \hat{T}'_0 &\leq C_1 n^3 (n\log n)^{1/4} + C_1 n^3 \log n, \\ \hat{T}''_k - \hat{T}'_{k-1} &\leq C_2 (2/3)^{k-1} n^3 (n\log n)^{1/4} + 1, \\ \hat{T}'_k - \hat{T}''_k &\leq C_3 (2/3)^{k/3} n^3 (n\log n)^{1/4} + C_3 n^3 \log n \end{aligned}$$

Plugging them into the telescopic sum we obtain the optimization time

$$\hat{T}'_{k*} = \hat{T}'_0 + \sum_{k=1}^{k^*} \left(\hat{T}'_k - \hat{T}''_k\right) + \sum_{k=1}^{k^*} \left(\hat{T}''_k - \hat{T}'_{k-1}\right). \tag{4.26}$$

If this event occurs, we can bound the summands of the optimization time from Equation (4.26) to

$$\sum_{k=1}^{k^*} (\hat{T}''_k - \hat{T}'_{k-1}) \leq \Big(\sum_{k=1}^{k^*} (2/3)^{k-1}\Big) C_2 n^3 (n\log n)^{1/4} + k^* \tag{4.27}$$

$$\sum_{k=1}^{k^*} (\hat{T}'_k - \hat{T}''_k) \leq \Big(\sum_{k=1}^{k^*} (2/3)^{k/3}\Big) C_3 n^3 (n\log n)^{1/4} + C_3 k^* n^3 \log n. \tag{4.28}$$

Substituting $k^* \leq 2\log n$ in the summands, upper bounding the geometric series, and inserting them into Equation (4.26) together with $\hat{T}'_0$ yields

$$\hat{T}'_{k*} \leq \underbrace{(C_1 + 3C_2 + 8C_3)}_{(*)} n^3 (n\log n)^{1/4} + \underbrace{C_1 n^3 \log n + 2\log n + 2C_3 n^3 (\log n)^2}_{(**)}.$$

Now, we can upperbound $(*)$ by $Dn^3(n\log n)^{1/4}$ and $(**)$ by $D'n^3(n\log n)^{1/4}$ for two absolute constants $D, D' > 0$, such that

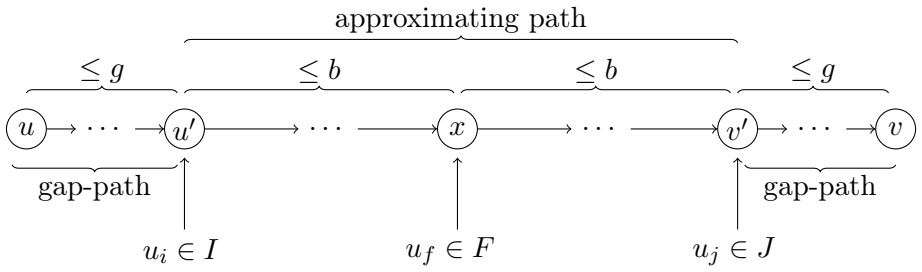


Figure 4.6: The path depicts the situation of Lemma 4.25 with two possible gap paths.

$$\hat{T}'_{k*} \leq (D + D')n^3(n \log n)^{1/4} \tag{4.29}$$

holds for all $n \in \mathbb{N}$. We apply a Union Bound argument similar to the previous section to prove Inequality (4.22) for constant $C := D + D'$. We take a closer look into the $2k^* + 1$ events considered in Equation (4.26) for which we had assumed that they would hold. If there exists a $k \in \{0, \ldots, k^*\}$ such that one of those event fails, then we get the events considered in (4.23) or (4.24) or (4.25) and thus the event considered in Equation (4.22). Summing up the failure probabilities given in Proposition 4.24 for at most n events yields Inequality (4.22) which implies Theorem 4.1. What is left now is to prove Proposition 4.24.

The two Inequalities (4.23) and (4.25) are based on the mutation operator only, hence they are a direct consequence of Lemma 4.20 from the previous section. We refrain from repeating a copy of the proof of Inequality (4.23) and (4.25) and remark that it works along the same lines. The only differences are due to a changed gap value $c(0)$ and $c(k)$ (Equality (4.19)) and hence the random variable $\hat{T}''_k$ which replaces t_0.

To prove Inequality (4.24) in Proposition 4.24 we need Lemma 4.25 providing a lower bound on the crossover success probability within the k-th crossover phase to produce a weight-minimal approximating path with gap at most $g(k)$. The gap is with respect to any weight-minimal path of length $c(k)$, and we assume that the previous $k-1$-th phase has been successfully finished and thus there exists a weight-minimal path (again of arbitrary length) in the population of Steady State $\mathrm{GA}_{\mathrm{APSP}}$ for every weight-minimal path of length at most $c(k-1)$.

> **Lemma 4.25.** *For every $b \in \mathbb{R}^+$ with $b \geq 24$, every $g \in \mathbb{R}^+$ with $g \leq b/8$, and every pair $(u, v) \in V^2_{(3/2)b} \setminus V^2_b$, there exists at least $bg^2 6^{-1}$ differ-*

> *ent tuples (u', x, v') of three distinct vertices $u', x, v' \in V$ such that the pairs (u', x), (x, v') are in V_b^2 and such that the concatenations of every weight-minimal u'-x-path and every weight-minimal x-v'-path at the vertex x results in a weight-minimal u'-v'-path which is an approximating path with gap at most g for the pair (u, v); compare Figure 4.6.*

Proof. Let the vertex pair (u, v) be in the set $V^2_{(3/2)b}$ but not in the set V_b^2. By the definitions of V_b^2 and $V^2_{(3/2)b}$, there exists a weight-minimal u-v-path $P_{u,v} = (u = u_0, u_1 \ldots, u_\ell = v)$ of length ℓ with

$$b < \ell \leq (3/2)b\,.$$

Consider the three index sets

$$\begin{aligned} I &:= \{0, \ldots, \lfloor g \rfloor\}, \\ F &:= \{\lceil b/2 \rceil, \ldots, \lceil b/2 \rceil + \lceil b/6 \rceil\}, \\ J &:= \{\ell - \lfloor g \rfloor, \ldots, \ell\}. \end{aligned}$$

of size $|I| = |J| \geq g$ and $|F| \geq b/6$ since $\ell > b$ and $g \leq b/8 \leq \lceil b/3 \rceil$ due to the assumption we make on g and b. Moreover, the index sets I, F, and J are pairwise disjoint subsets of $\{0, \ldots, \ell\}$ for every choice of ℓ, g, and b. We choose $i \in I$, $f \in F$, and $j \in J$ and the vertices $u' := u_i$, $x := u_f$, and $v' := u_j$. Then, by an application of Lemma 4.9 and Corollary 4.10, the two paths $P_{u',x} = (u' = u_i, \ldots, u_f = x)$, and $P_{x,v'} = (x = u_f, \ldots, u_j = v')$ are weight-minimal, too. Moreover, both paths are of length at most b, hence, (u', x), and (x, v') are in V_b^2. By the application of Lemma 4.9, the result of the concatenation of the two weight-minimal paths $P'_{u',x}$, and $P'_{x,v'}$ (of arbitrary length) at the vertex x is weight-minimal, too. Finally, there are

$$|I| \cdot |F| \cdot |J| = \frac{g^2 \cdot b}{6}$$

ways to choose the tuple (u', x, v'), which concludes the proof of this lemma. □

With this lemma at hand we are able to prove Inequality (4.24) which also finishes the proof of Proposition 4.24.

Proof of Inequality (4.24). We need to show that for every $k \in \{1, \ldots, k^*\}$ at most $C_2(2/3)^{k-1}n^3(n \log n)^{1/4} + 1$ iterations (for a suitable constant C_2) of the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ suffice such that the k-th crossover phase which starts

after the preceding stage $\hat{T}'_k$ finishes successfully. The crossover phase fails with probability at most $1/n^2$ as stated in Inequality (4.24). This is sufficient because we will sum up at most n such events in the Union Bound later. Time $\hat{T}'_{k-1}$ marks the beginning of the crossover phase and we assume that the population of the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ contains a weight-optimal path for every pair of vertices contained in $V^2_{c(k-1)}$.

We use Lemma 4.12 (Coupon Collector with dependencies) in combination with Lemma 4.25 to collect a weight-optimal approximating path with gap at most $g(k)$ for every pair of vertices from $I := V^2_{c(k)} \backslash V^2_{c(k-1)}$ at time $\hat{T}'_{k-1}+t+1$ in the population of Steady State $\mathrm{GA}_{\mathrm{APSP}}$. Denote this event by $A^t_{(u,v)}$ with $t \in \mathbb{N}$ and $(u,v) \in I$ and we show for every $t \in N$ and $(u,v) \in I$

$$\mathbf{Pr}\left[A^t_{(u,v)} \,\middle|\, \bigcap_{0 \le s < t} \overline{A^s_{(u,v)}}\right] \ge p.$$

Due to the definition of I there is no $(u,v) \in V^2_{c(k-1)}$ in the population of the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ at time $\hat{T}'_{k-1}+t$ with positive probability and thus the conditional probability is well-defined. Since we assumed that there is a weight-optimal path for pair of vertices in $V^2_{c(k-1)}$ we can use Lemma 4.25 to deduce probability of at least $g(k)^2 \cdot b/6$ to create a weight-minimal approximating path with gap at most $g(k)$ for an arbitrary pair of vertices in $(u,v) \in I$. This is achieved by the concatenation of two weight-optimal pairs (u',x) and (x,v') from $V^2_{c(k-1)}$ yielding a weight-minimal path (u',v') which approximates weigh-minimal path (u,v) with gap at most $g(k)$. Due to the restrictions imposed on the length of the gap that needs to be closed by mutation we again need here $c(k-1) \ge 24$ which trivially holds because $c(k-1) \ge c(0)$ is monotonically increasing and $c(0) := 24(n \log n)^{1/4}$. The second assumption in Lemma 4.25 states that $g(k) \le c(k-1)/12$ which also holds since $g(k)$ is monotonically decreasing, $c(k)$ is monotonically increasing with k and $g(0) \le c(0)$.

The Steady State $\mathrm{GA}_{\mathrm{APSP}}$ performs this particular concatenation at time $\hat{T}'_{k-1}+1+t$ if (i) the algorithms performs a crossover step, (ii) chooses the weight-minimal u'-x-path as first parent, and (iii) then the weight-minimal x-v'-path as second parent from the population at time $\hat{T}'_{k-1}+t$. The event (i) has probability p_c. By the definition of the basic crossover operator (Algorithm 4.3) event (ii) happens with probability at least $1/n^2$ and event (iii) happens again with probability at least $1/n^2$. These bounds are independent of the event whether or not there already exists a weight-minimal approximating path of gap at most g for the pair (u,v) in the population at time $\hat{T}'_{k-1}+t$.

Equation (4.24) indeed holds for all $(u,v) \in I$ and $t \in \mathbb{N}$ with

$$p := \frac{c(k-1)g(k)^2 p_c}{6n^4} = \frac{4(3/2)^{k-1} \cdot (2/3)^{2k/3}(n \log n)^{3/4} p_c}{n^4}.$$

Finally, since the event that at time $\hat{T}'_{k-1} + 1 + t$ one of the events $A^0_{(u,v)}, \ldots, A^t_{(u,v)}$ has occurred for all $(u,v) \in I$ implies the event "$\hat{T}''_k \leq \hat{T}'_{k-1} + 1 + t$", we have by Lemma 4.12 (Coupon Collector with Dependencies) with $r := C_2(2/3)^{k-1} n^3 (n \log n)^{1/4}$ and $C_2 := 1/p_c$ that

$$\begin{aligned} pr &= \frac{4(3/2)^{k-1}(2/3)^{2k/3}(n \log n)^{3/4} p_c}{n^4} \cdot (2/3)^{k-1} n^3 (n \log n)^{1/4} \cdot \frac{1}{p_c} \\ &\geq 4 \log n \geq 4 \ln n \end{aligned}$$

and therefore, since $I \leq n^2$ that

$$\mathbf{Pr}\left[\hat{T}''_k \geq \hat{T}'_{k-1} + 1 + C_2(2/3)^{k-1}(3/2)^{2k/3} n^3 (n \log n)^{1/4}\right] \leq |I| \mathrm{e}^{-4 \ln n} \leq \frac{1}{n^2}.$$

This, again, concludes the proof of Proposition 4.24 and therefore also the proof of Theorem 4.1. □

4.7 Multi-criteria Shortest Paths

The multi-criteria shortest path problem is one of the most fundamental problems in computer science with high relevance in practice. These types of problems easily arise in train networks [28] if for example the criteria "travel time", "number of transfers" as well as "reliability of transfers" are considered.

In this section we discuss a genetic algorithm for the problem based on publication [111], co-authored with Frank Neumann. In the original publication we presented a genetic algorithm stated here as Algorithm 4.5 employing our simple crossover introduced in Section 4.4. Here, we present a new and improved and so far unpublished analysis of the multi-objective shortest path problem following the ideas of Section 4.5.2. By this we make some argument precise that had to be left out from [111] due to space restrictions.

4.7.1 Statement of the Optimization Problem

Similar to the single-criteria APSP considered in Section 4.3 the input is a connected directed graph $G = (V, E)$ with n vertices and m edges, and a positive and normalized weight function $w\colon E \to (\mathbb{R}^+_{\geq 1})^d$ which assigns to each edge a vector with $d \geq 2$ weights. We denote by $w_{\max} = \max_{i=1}^{d}(\max_{e \in E} w_i(e))$ the maximum weight of the given input.

Similar to the notation introduced in Section 2.2.1 we need a partial order in order to define Pareto dominance as well as our optimization goal. We define the following notions with respect to the multi-objective problem at hand.

Due to the definition of APSP we only consider paths to be comparable with respect to Pareto dominance that have the same start and end vertex. Here, we define the dominance relation in the objective space.

Definition 4.26 (Pareto dominance)**.** *Let $P_{u,v}$ be a walk from u to v in G and*

$$w(P_{u,v}) = (w_1(P_{u,v}), \ldots, w_d(P_{u,v}))$$

the corresponding objective vector which gives the different paths weights with respect to the d objective functions. Comparing two walks $P_{u,v}$ and $P'_{u,v}$, we write $w(P_{u,v}) \preceq w(P'_{u,v})$ iff $w_i(P_{u,v}) \leq w_i(P'_{u,v})$, $1 \leq i \leq d$. Similarly, we write $w(P_{u,v}) \prec w(P'_{u,v})$ iff $w(P_{u,v}) \preceq w(P'_{u,v})$ and $w_i(P_{u,v}) < w_i(P'_{u,v})$ for at least one i.

Considering a multi-objective problem, one is interested in computing or approximating the Pareto front.

Definition 4.27 (Pareto optimality)**.** *$\hat{P}_{u,v}$ is called Pareto optimal if there is no other path $\hat{P}'_{u,v} \neq \hat{P}_{u,v}$ from u to v for which $w(\hat{P}'_{u,v}) \prec w(\hat{P}_{u,v})$ holds.*

Definition 4.28 (Pareto set)**.** *Let $\mathcal{S}$ denote the set of all feasible instances, then the Pareto set $\mathbb{P}_{u,v} \subseteq \mathcal{S}$ for a pair of distinct vertices u, v is the set of all Pareto optimal paths $\hat{P}_{u,v}$, i. e. $\mathbb{P}_{u,v} := \{\hat{P}_{u,v} \mid \hat{P}_{u,v} \in \mathcal{S}$ and $\hat{P}_{u,v}$ is Pareto optimal$\}$. The whole Pareto set is given by $\mathbb{P} := \bigcup_{u,v \in V} \mathbb{P}_{u,v}$*

Mapping $\mathbb{P}$ to the objective space defines the Pareto front. Throughout this section we will mainly deal with Pareto sets and make use of the Pareto front with a notion of approximation.

Definition 4.29 (Pareto front)**.** *Let $\mathcal{S}$ denote the set of all feasible instances and $w(\mathcal{S})$ the objective space. The Pareto front $\mathcal{F}_{u,v} \subseteq w(\mathcal{S})$ for a given pair of distinct vertices u, v is the set of all objective vectors for which a Pareto optimal path from u to v exists. The whole Pareto front $\mathcal{F} \subseteq \mathcal{S}$ for the multi-criteria APSP constitutes of the Pareto fronts $\mathcal{F}_{u,v}$ for each distinct pair of vertices u and v, i. e., $\mathcal{F} := \bigcup_{u,v \in V} \mathcal{F}_{u,v}$.*

Computing the Pareto front for the multi-criteria APSP is NP-hard if and only if $d \geq 2$ (see [54, Chapter 9.1])[5]. On the other hand, the number of Pareto optimal objective vectors might grow exponentially with respect to the given input. It is also known for the multi-objective shortest path problem that there are instances which have exponentially many Pareto optimal paths with respect to the input size, compare [54, Chapter 9.1] Due to this, we are interested in an approximate solution for this problem. In this case, it is only necessary to compute a smaller set of solutions, for which we introduce a notion of approximation in the objective space.

For the single-criteria case the definition of an approximation algorithm is as follows.

Definition 4.30 (Approximation algorithm)**.** *Let A be an approximation algorithm for a minimization problem with ratio $\rho > 0$. This means that for any input instance x algorithm A computes a solution $A(x)$ whose value $f(A(x))$ will not be more than ρ times the optimal value $\mathrm{OPT}(x)$ of a solution to x, i. e. $f(A(x)) \leq \rho \, \mathrm{OPT}(x)$.*

For the multi-criteria case we need to extend this notion to d objectives.

Definition 4.31 (ε-dominance, ε-approximation)**.** *Let $\mathcal{S}$ denote set of all feasible instances and $w(\mathcal{S})$ the objective space. Then $y \in w(\mathcal{S})$ ε-dominates $y' \in w(\mathcal{S})$ denoted by $y \preceq_\varepsilon y'$ iff $w_i(y) \leq \varepsilon \cdot w_i(y')$ for all $1 \leq i \leq d$. We say that a set $\mathcal{P} \subseteq \mathcal{S}$ ε-approximates set $\mathcal{P}' \subseteq \mathcal{S}$ iff for every $y' \in w(\mathcal{P}')$ there is a $y \in w(\mathcal{P})$ such that $y \preceq_\varepsilon y'$.*

For the Pareto front this means that a set $\mathcal{P} \subseteq \mathcal{S}$ ε-approximates the Pareto front $\mathcal{F}_{u,v}$ or even the whole Pareto front $\mathcal{F}$ if and only if for every $y \in \mathcal{F}_{u,v}$ or $y \in \mathcal{F}$ there is a $P_{u,v} \in \mathcal{P}$ such that $w(P_{u,v}) \preceq_\varepsilon y$.

[5] It is NP-hard to determine for two objectives whether a point belongs to the Pareto front. We can reduce Knapsack to the two-dimensional problem if we optimize just one objective while keeping the other one fixed.

Our goal is to compute for each pair of distinct vertices u and v and each Pareto optimal path $P_{u,v}$ an $(1+\varepsilon)$-approximation $P'_{u,v}$ of $P_{u,v}$, i. e., a path $P'_{u,v}$ for which $w(P'_{u,v}) \leq (1+\varepsilon) \cdot w(P_{u,v})$ holds. Here $\varepsilon > 0$ is a parameter that determines the quality of the approximation.

This notion of a multiplicative $(1+\varepsilon)$-approximation was independently introduced in the 80's by several authors [57, 69, 100, 130, 135]. It relates well to the concept of ε-dominance used in evolutionary multi-objective optimization [93, Chapter 4.3].

> **Definition 4.32** (FPRAS)**.** *An algorithm for an optimization problem is called fully-polynomial time randomized approximation scheme (FPRAS) if it runs in polynomial time w. r. t. the size of the input and $1/\varepsilon$, and outputs an $(1+\varepsilon)$-approximate Pareto set with probability at least $3/4$ for any input instance and any $\varepsilon \in \mathbb{R}^+$.*

An FPTAS is the deterministic version of an FPRAS, that is the probability to output an $(1+\varepsilon)$-approximate Pareto set with probability 1.

Despite its practical importance it took until FOCS 2000 when Papadimitriou and Yannakakis [122] gave a first theoretical approximation guarantee of the Pareto front. This was later extended by different authors [25, 26, 121, 155]. In [151, 152], Tsaggouris and Zaroliagis give the best-known FPTAS for the multi-criteria single-source shortest path problem. For the same problem [75] states an evolutionary approach for which the author presents a FPRAS (fully-polynomial time randomized approximation scheme) based on [151, 152].

The concept of ε-dominance is implemented in the algorithm in terms of hyperboxing (dividing the objective space into hyperboxes) and box-domination which was to the best of our knowledge first introduced by [151, 152] but generalized and introduced into evolutionary computation by [93].

> **Definition 4.33** (Hyperbox Index Vector)**.** *Let $r > 1$ be a parameter determining the size of the hyperboxes. We assume that the objective space is positive and normalized. Then the box index vector $b_r(P_{u,v})$ of path $P_{u,v}$ is given by*
>
> $$b_r(P_{u,v}) := (\lfloor \log_r(w_1(P_{u,v})) \rfloor, \lfloor \log_r(w_2(P_{u,v})) \rfloor, \ldots, \lfloor \log_r(w_d(P_{u,v})) \rfloor).$$

This means that we place a hyper-grid in the objective space with coordinates $r, r^2, r^3, \ldots$ for each objective, compare Figure 4.7. We will see later in Equation (4.36) how r and ε mutually depend on each other.

Horoba and Neumann [76] discuss the benefits and drawbacks for the use of ε-dominance in combination with hyperboxing. One of the obvious disadvantages of this concept is that the ε is fixed in advance and its inability to dynamically adapt it to the problem at hand. To this end, Bringmann et al. [15] propose a practical algorithm that aims at improving the approximation of the Pareto front over time.

4.7.2 Algorithm

For the multi-criteria APSP we investigate an algorithm called Diversity Evolutionary Multi-objective Optimizer (DEMO), which is an extension of the population-based approach for the single-criteria APSP discussed in the previous section. In its population it only keeps individuals from non-dominated hyperboxes.

Let $\mathcal{S}$ be the *search space* of all walks up to length n. Each individual $P_{u,v}$ in the population taken from the search space is a path from u to v, where u and v are two distinct vertices of the given input graph. The fitness of $P_{u,v}$ is given by $w(P_{u,v})$. Our algorithm starts with a population $\mathcal{P} := \{P_{u,v} = (u, v) \mid (u, v) \in E\}$ of size $|E|$ which contains all paths corresponding to the edges of the given graph G. In each iteration a single new individual is produced either by recombination or mutation depending on the pre-defined probability p_c. For all investigations in this paper, we assume that p_c is chosen as an arbitrary constant, i. e. $p_c \in]0, 1[$ and $p_c = \Omega(1)$. Our **goal** is to evolve an initial population consisting of a set of paths into a population which approximates the Pareto-front with respect to an arbitrary factor ε chosen by the user. We will generally assume that $\log(w_{\max})$ and $1/\varepsilon$ are polynomially bounded w. r. t. n.

We use the *mutation operator* defined in Algorithm 4.2 which simulates the classical mutation operator on bits that flips each bit with probability $1/n$.

For the choice of the *crossover operator* we have some freedom. We could choose our simple crossover from Section 4.4 or we could decide to use the repair crossover from Section 4.5.1. Contrary to the mutation operator a new individual created by such a crossover operation may no longer represent a path and thus constitute an invalid solution which is rejected by the algorithm. For this reason we will use the feasible parent selection stated in Algorithm 4.4 in Section 4.5.2.

Finally the *environmental selection* of Line 9 and 10 includes such a new individual in the population if and only if it is not dominated by some other s-t-path. Inserting the new individual in the population all other s-t-paths

Algorithm 4.5: DEMO(r) (Diversity Evolutionary Multi-objective Optimizer with parameter r which defines the sizes of the hyperboxes)

Input: Parameter r to scale the size of the hyperboxes (cf.Def. 4.33);
$\mathcal{P} \leftarrow \{P_{u,v} = (u, v) \mid (u, v) \in E\}$;
while *true* **do**
  Choose $c \in [0, 1]$ uniformly at random;
  if $c \leq p_c$ **then**
    Choose two individuals $P_{x,y}$ and $P_{x',y'}$ from $\mathcal{P}$ u. a. r.;
    Perform CROSSOVER on $P_{x,y}$ and $P_{x',y'}$ to obtain an individual $P'_{s,t}$;
  else
    Choose one individual $P_{x,y}$ uniformly at random from $\mathcal{P}$ and perform MUTATION on $P_{x,y}$ to obtain an individual $P'_{s,t}$;
  Perform SELECTION: **if** *($P'_{s,t}$ is a path from s to t) and (there is no $P''_{s,t} \in \mathcal{P}$ with $w(P''_{s,t}) \leq w(P'_{s,t})$ and $w(P''_{s,t}) \neq w(P'_{s,t})$)* **then**
    exclude all $P''_{s,t}$ where $b_r(P'_{s,t}) \leq b_r(P''_{s,t})$ and add $P'_{s,t}$ to $\mathcal{P}$;

residing in the same hyperbox are removed. Here, it is sufficient to have one individual per hyperbox which is not dominated by other occupied hyerboxes in the population which is illustrated in Figure 4.7. Assuming that the multi-objective optimization problem is a pure minimization problem then any individual which we insert into the population r-dominates other individuals in the gray shaded area.

After we discussed the notion of hyperboxes and their use with the concept of box domination within the Algorithm DEMO the question arises how large the population might become. In contrast to the single-criteria APSP this is not immediately clear. For every hyperbox that the algorithm encounters which is not dominated we need to keep at most an individual for every pair of vertices. In the following Lemma which consequence of Theorem 2 given in [93] we upper bound the population size of DEMO by upperbounding the number of non-dominated hyperboxes.

Lemma 4.34. *Let $r > 1$ be the input of* DEMO(r)*, $w_{\max}$ be the maximum weight of the weight function w, and $\mathcal{P}_{\max}$ be the maximal population size. Then*

$$\mathcal{P}_{\max} \leq n(n-1) \cdot \left(\frac{\log(n \cdot w_{\max})}{\log(r)} \right)^{d-1}.$$

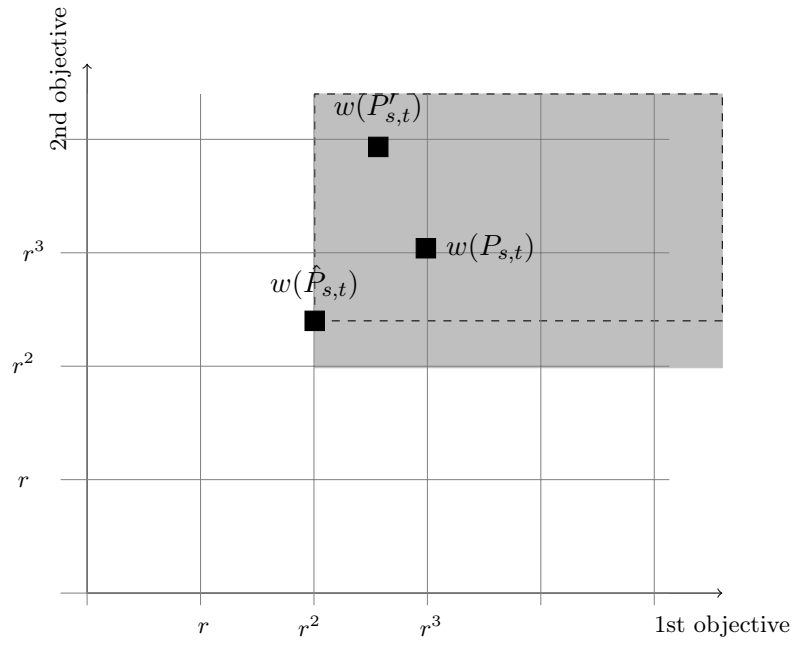


Figure 4.7: Let $\hat{P}_{s,t} \in \mathbb{P}$ be a Pareto-optimal path for the vertex pair s and t. Then $\hat{P}_{s,t}$ Pareto-dominates all individuals that are within the dashed area. On the other hand $P_{s,t}$ r-dominates all individuals within the lightgray area. I. e. $w(P_{s,t})$ r-approximates $w(\hat{P}_{s,t})$. Assume that at some point in time $P'_{s,t}$ which resides in the same hyperbox as $P_{s,t}$ enters the population. The environmental selection of DEMO replaces $P_{s,t}$ by $P'_{s,t}$. Then due to the size of the hyperbox our approximation of the Pareto-optimal $\hat{P}_{s,t}$ gets worse and $w(P'_{s,t}) \leq r^2 \cdot w(\hat{P}_{s,t})$ holds.

Proof. The objective space in each dimension extends at most to $(n \cdot w_{\max})$, that is it is divided into at most $\left(\frac{\log(n \cdot w_{\max})}{\log(r)}\right)^d$ many hyperboxes. As we only keep individuals from non-dominated hyperbox, we easily see that there are at most $\left(\frac{\log(n \cdot w_{\max})}{\log(r)}\right)^{d-1}$ equivalence classes where at least one coordinate differs. In each non-dominated hyperbox we keep at most one individual for each distinct tuple of vertices (u, v). □

We study the runtime of DEMO until it has produced an $(1+\varepsilon)$-approximation of the multi-criteria APSP. The runtime is again measured by the optimization time, the number of iterations (of the while loop) until the algorithm has achieved the desired approximation.

Theorem 4.35. *Let $(1+\varepsilon)$ be the desired approximation ratio for* DEMO *and $r = (1+\varepsilon)^{1/(2\log n)}$ be the input parameter of* DEMO*. Then for every crossover rate $p_c \in (0,1)$ there exists an absolute positive constant $C := C(p)$ such that* DEMO(r) *w. h. p. computes an $(1+\varepsilon)$-approximation for the Pareto front of the multi-criteria APSP in optimization time $C \cdot \mathcal{P}_{\max} \cdot n \log n$.*

The bound given in Theorem 4.35 depends on the approximation factor $(1+\varepsilon)$ and the size of the hyperboxes $r = (1+\varepsilon)^{1/(2\log n)}$. (This relation is explained in Equation (4.36) in the proof of the main theorem.) Furthermore, the maximal population size $\mathcal{P}_{\max}$ that DEMO might encounter plays an important role. The bound on $\mathcal{P}_{\max}$ given in Lemma 4.34 depends on the number of given objectives as well as on the maximal weight $w_{\max}$ and the approximation guarantee $\log(r)$ both of which are required to be polynomially bounded in n.

Since

$$\begin{aligned}\mathcal{P}_{\max} &\leq n^2 \cdot \left(\frac{\log(n \cdot w_{\max})}{\log(r)}\right)^{d-1} \\ &\leq n^2 \cdot (2\log n)^{d-1} \left(\frac{\log(n \cdot w_{\max})}{\log(1+\varepsilon)}\right)^{d-1}, \end{aligned} \tag{4.30}$$

the optimization time is upper bounded by

$$C \cdot \mathcal{P}_{\max} \cdot n \log n \leq C \cdot n^3 \cdot (2\log n)^d \cdot \left(\frac{\log(n \cdot w_{\max})}{\log(1+\varepsilon)}\right)^{d-1}$$

w. h. p. such that we conclude that DEMO is an FPRAS.

Assuming that instead of the feasible parent selection the simple crossover of Section 4.4 is introduced in Algorithm DEMO (Line 7), then we would get the following bound (without proof) on the optimization time of DEMO.

Corollary 4.36. *Let $(1+\varepsilon)$ be the desired approximation ratio for* DEMO*. Let $g := \left(\frac{\mathcal{P}_{\max} \log n}{n}\right)^{1/4}$, and $r = (1+\varepsilon)^{1/(3 \cdot g \log n)}$ be the input parameter of* DEMO*. Then for every crossover rate $p_c \in (0,1)$ there exists an absolute positive constant $C := C(p)$ such that* DEMO(r) *with the simple crossover w. h. p. computes an $(1+\varepsilon)$-approximation for the Pareto front of the multi-criteria APSP in optimization time $Cn \cdot \mathcal{P}_{\max} \cdot g$.*

Inserting the repair operator discussed in Section 4.5.1 we get the following bound (without proof) on the optimization time.

Corollary 4.37. *Let $(1+\varepsilon)$ be the desired approximation ratio for* DEMO*. Let $g' := \left(\frac{\mathcal{P}_{\max} \log n}{n}\right)^{1/5}$, and $r = (1+\varepsilon)^{1/(3 \cdot g' \log n)}$ be the input parameter of* DEMO*. Then for every crossover rate $p_c \in (0,1)$ there exists an absolute positive constant $C := C(p)$ such that* DEMO(r) *with the repair crossover w. h. p. computes an $(1+\varepsilon)$-approximation for the Pareto front of the multi-criteria APSP in optimization time $Cn \cdot \mathcal{P}_{\max} \cdot g'$.*

We discuss the interesting cases for the value of d. Setting $d := 1$ in Corollary 4.37 we get the well-known tight bounds of $\Theta\left(n^{3.25}\right)$ on the optimization time for the single-criteria case from Section 4.4. Note, that no approximation is needed in this case. For Corollary 4.36 and $d = 1$ we get the bounds of the single-criteria case discussed in Section 4.5.1, and finally for our main Theorem 4.35 and $d = 1$ we get the same bounds as discussed in Section 4.5.2 as expected.

For dimensions $d > 1$ the population size and thus the optimization time is influenced not only be the dimension but also by the approximation factor $(1+\varepsilon)$ we would like to guarantee.

The population size $\mathcal{P}'_{\max}$ for the simple crossover turns into

$$\mathcal{P}'_{\max} \leq n^{\frac{9-d}{5-d}} \cdot (\log n)^{\frac{5(d-1)}{5-d}} \cdot \left(\frac{3\log(nw_{\max})}{\log(1+\varepsilon)}\right)^{\frac{4(d-1)}{5-d}},$$

for which we can go up to dimension $d = 4$.

And the population size $\mathcal{P}''_{\max}$ for the repair crossover is upper bounded by

$$\mathcal{P}''_{\max} \leq n^{\frac{11-d}{6-d}} \cdot (\log n)^{\frac{6(d-1)}{6-d}} \cdot \left(\frac{3\log(nw_{\max})}{\log(1+\varepsilon)}\right)^{\frac{5(d-1)}{6-d}},$$

for dimensions $d \leq 5$.

Now, setting the dimension d to a constant k (suitably bounded according to the discussion in the previous lines), the corresponding worst-case bounds are:

For feasible parent selection

$$O\left(n^3 (\log n)^k \left(\frac{2\log(nw_{\max})}{\log(1+\varepsilon)}\right)^{k-1}\right),$$

for repair crossover

$$O\left(n^{\frac{18-2k}{6-k}} \cdot (\log n)^{\frac{7k+1}{6-k}} \cdot \left(\frac{3\log(nw_{\max})}{\log(1+\varepsilon)}\right)^{\frac{6(k-1)}{6-k}}\right),$$

and for simple crossover

$$O\left(n^{\frac{15-2k}{5-k}} \cdot (\log n)^{\frac{6k-5}{5-k}} \cdot \left(\frac{3\log(nw_{\max})}{\log(1+\varepsilon)}\right)^{\frac{5(k-1)}{5-k}}\right)$$

Interestingly for feasible parent selection the worst-case optimization time depends on the dimension in a negligible fashion only.

4.7.3 Analysis

For the analysis of DEMO we need to consider a slightly adapted notion of Pareto-optimality which also takes into account the length of the path. Among all paths of length at most k we define an s-t-path $\hat{P}^k_{s,t}$ to be Pareto-optimal with respect to length k if there is no other path $\tilde{P}^k_{s,t}$ of length at most k for which $w(\tilde{P}^k_{s,t}) < w(\hat{P}^k_{s,t})$ holds.

Definition 4.38. *Let $G = (V, E)$ be a graph and let $k \in \mathbb{R}$. We denote by $\mathbb{P}_k$ the set of Pareto-optimal paths of length at most k:*

$$\mathbb{P}_k = \{P_{u,v} \mid (u,v) \in V^2 \wedge |P_{u,v}| \leq k \wedge \nexists \tilde{P}_{u,v} \colon |\tilde{P}_{u,v}| \leq k \wedge \tilde{P}_{u,v} \prec P_{u,v}\}.$$

Remark 4.39. *Let us map the objective vectors of the set $\mathbb{P}_k$ into the set of hyperboxes $\mathbb{B}_k := b_r(w(\mathbb{P}_k))$. We call a hyperbox $B \in \mathbb{B}_k$ dominated with respect to a pair of vertices $(u,v) \in V^2$ if and only if there is an individual $P_{u,v} \in \mathcal{P}$ such that $b_r(w(P_{u,v})) \prec B$.*

With the concept of box-domination we keep the size of the population as small as claimed in Lemma 4.34 because we only keep individuals from non-dominated hyperboxes in the population. Furthermore, for each non-dominated hyperbox and every s-t-path, $s, t \in V$, we have at most one individual. At the same time we are able to control the approximation-factor of the individuals in the population.

Assume that for every Pareto-optimal path

$$\hat{P}_{s,t} = (s = v_0, v_1, \ldots, v_k = v, v_{k+1} = t) \in \mathbb{P}_{k+1}$$

there is an individual $P_{s,v}$ in our population that r^i-approximates $\hat{P}_{s,v} \in \mathbb{P}_k$. Then it is clear that there is a mutation operation which appends the edge (v, t) to $P_{s,v}$. This increases the length of $P_{s,v}$ by one and results in an individual $P_{s,t}$ that r^i-approximates $\hat{P}_{s,t}$. During the remaining process of the optimization it might happen that we replace the path $P_{s,t}$ by some path $P'_{s,t}$ that is not dominated by $P_{s,t}$ and resides in the same hyperbox. In such a situation the approximation factor may increase from r^i to at most r^{i+1} while at the same time $P'_{s,t}$ also r^{i+1}-approximates all Pareto-optimal paths from s to t mapped to the hyperbox $b_r(w(\hat{P}_{s,t}))$ (see Lemma 4.40 and Figure 4.7).

Lemma 4.40 (Based on Lemma 1 of Horoba [75])**.** *Let $\hat{P}_{s,t}$ be an arbitrary path of the search space $\mathcal{S}$. Assume that there is a path $P_{s,t} \in \mathcal{P}$ in the current population with $w(P_{s,t}) \preceq_{r^i} w(\hat{P}_{s,t})$. Then all subsequent populations $\mathcal{P}'$ of* DEMO(r) *with $r > 1$ fulfill*

$$w(\mathcal{P}') \preceq_{r^{i+1}} \cdot w(P'_{s,t})$$

for all $P'_{s,t} \in \mathcal{S}$ with $b_r(w(P'_{s,t})) \preceq b_r(w(\hat{P}_{s,t}))$.

We would now like to analyze the algorithm in stages, whereas we call the k-th stage completed for graph G if the population $\mathcal{P}$ r^k-approximates the Pareto set $\mathbb{P}_{d(k)}$ for the sequence $\{d(k)\}_{k \in \mathbb{N}}$ given by

$$d(k) := \left(\frac{3}{2}\right)^k \tag{4.31}$$

The first point in time when this property holds defines the random variable $\bar{T}_k$ which then marks the end of the k-stage.

Definition 4.41 (Time $\bar{T}_k$)**.** *For $k \in N$ let $d(k)$ be defined as in Equation 4.31. Furthermore, let $\bar{T}_k$ denote the first point in time such that for all paths in $\mathbb{P}_{d(k)}$ the population $\mathcal{P}$ of* DEMO *is an r^k-approximation of $\mathbb{P}_{d(k)}$.*

Opposed to the proofs in the previous sections we cannot guarantee that our population contains the Pareto optimal paths we are looking for. As such we

restrict to the notion of approximation used in Definition 4.41. It does not matter what the exact length of the paths in the population are that achieve this approximation.

By Definition 4.41 it is clear that $\bar{T}_0 \leq \bar{T}_1 \leq \bar{T}_2 \leq \dots$ and some of the inequalities may happen to be tight. The discussion preceding Lemma 4.40 has made clear that the number of stages is crucial for the approximation factor achieved by the algorithm as in each stage the approximation factor will get worse by an additional r. Lemma 4.40 has made clear that the number of stages is crucial for the approximation factor achieved by the algorithm as in each stage the approximation factor will get worse by an additional r. The Pareto set $\mathbb{P}_{n-1}$ contains all paths that constitute the Pareto front of our problem. As a consequence the optimization time is dominated by such a k^*, that $d(k^*) \geq n-1$, which is certainly true for

$$k^* := \lceil \log_{1.5}(n) \rceil \leq 2 \log n \tag{4.32}$$

Thus, we deduce that the population of DEMO contains an $r^{2 \log n}$-approximation of the Pareto front after k^* stages at time $\bar{T}_{k^*}$.

Theorem 4.35 directly follows from the following inequality if we can prove that for every crossover rate $p_c \in \{0, 1\}$, there exists an absolute, positive constant C such that

$$\mathbf{Pr}\left[\bar{T}_{k^*} \geq C \cdot \mathcal{P}_{\max} \cdot n \log n\right] \leq \frac{1}{n} \tag{4.33}$$

holds. We remark, that in the definition of $\bar{T}_k$ the other input parameter r of DEMO which controls the size of the hyperboxes is included and needs to be treated within the remainder of this section.

There is a stronger statement to be made about the stages of the algorithm displayed in the following proposition.

Proposition 4.42. *For every crossover rate $p_c \in (0, 1)$ there exists an absolute, positive constant $C_1 := C_1(p)$ depending on p_c such that*

$$\mathbf{Pr}\left[\bar{T}_k \geq \bar{T}_{k-1} + C_1 \left(\frac{2}{3}\right)^k \cdot \mathcal{P}_{\max} \cdot n \log(n) + 1\right] \leq \frac{1}{n^2} \tag{4.34}$$

holds for all $k \in \{1, \dots, k^*\}$.

Inequality (4.33) follows from Inequality (4.34) by considering the telescopic sum

$$\bar{T}_{k^*} = \bar{T}_0 + \sum_{i=1}^{k^*} \bar{T}_k - \bar{T}_{k-1}.$$

For the random variable $\bar{T}_0$ we have $\bar{T}_0 := 0$ because by definition of $\bar{T}$ the populations needs to contain an r^0-approximating path for every Pareto optimal path from the set $\mathbb{P}_{d(0)}$. This is of course trivially given due to the initialization in Line 2 of DEMO. The population is initialized by the direct edges for every pair of vertices which at the same time are all Pareto optimal paths of length 1 from the set $\mathbb{P}_1$. Suppose that the statement $\bar{T}_k - \bar{T}_{k-1} \leq C_1(2/3)^k \cdot \mathcal{P}_{\max} \cdot n\log(n) + 1$ from Inequality (4.34) holds for every $k \in \{1, \ldots, k^*\}$, which we denote by event E. Inserting it into the telescopic sum, bounding $\sum_{i=1}^{k^*}(2/3)^k \leq \sum_{i=1}^{\infty}(2/3)^k$ by the geometric series and setting $k^* := 2\log(n)$ as given above, yields

$$\bar{T}_{k^*} \leq 2C_1\mathcal{P}_{\max} n\log(n) + 2\log(n).$$

For $C := 2C_1 + 1$ we deduce from event E the event $\bar{T}_{k^*} \leq C\mathcal{P}_{\max} n\log n$ and denote its converse as event A defined as $\bar{T}_{k^*} \geq C\mathcal{P}_{\max} n\log n$. For this event we would now like to show the error bound given in Inequality (4.33) assuming that Proposition 4.42 holds. If event A holds then this implies the event $\exists k \in \{1, \ldots, k^*\}\colon \bar{T}_k - \bar{T}_{k-1} \geq C_1(2/3)^k \cdot \mathcal{P}_{\max} \cdot n\log(n) + 1$ which is the union of the events E_k defined as $\bar{T}_k - \bar{T}_{k-1} \geq C_1(2/3)^k \cdot \mathcal{P}_{\max} \cdot n\log(n) + 1$, i. e. $\bigcup_{k=1}^{k^*} E_k$.

We apply the Union Bound (cf. Theorem 4.7) to at most $2\log n$ events which have probability at most $1/n^2$:

$$\begin{aligned}
\mathbf{Pr}\left[\bar{T}_{k^*} \geq C\mathcal{P}_{\max} n\log n\right] &\leq \mathbf{Pr}\left[\exists k \in \{1, \ldots, k^*\}\colon E_k\right] \\
&\leq \mathbf{Pr}\left[\bigcup_{k=1}^{k^*} E_k\right] \overset{4.7}{\leq} \sum_{i=1}^{k^*} \mathbf{Pr}\left[E_k\right] \\
&\leq \frac{1}{n}
\end{aligned}$$

After we attributed the proof of Theorem 4.35 to Proposition 4.42 we devote the remainder of this section to its proof. We will restrict to the potential of the crossover operator and in the proof neglect the positive effects of mutation. We will see that crossover is fast enough to create paths of increasing length

that approximate the appropriate Pareto paths. The proof follows the ideas that we have already seen in the previous section. We will assume that given a successful completion of stage k at time $\bar{T}_k$ we complete stage $k+1$ sufficiently fast. Assuming that at time $\bar{T}_k$ the population r^k-approximates the Pareto set $\mathbb{P}_{d(k)}$ with $d(k) = (3/2)^k$ it is possible to get an r^{k+1}-approximating path of the Pareto set $\mathbb{P}_{d(k+1)}$ with $d(k+1) = 3/2 \cdot d(k)$ into the population with sufficiently good probability. Note, that we only keep paths in the population that stem from non-dominated hyperboxes. Consequently, we restrict to the creation of those paths only fall within non-dominated hyperboxes. The precise length the paths within our population is not important, we only guarantee the approximation factor. Within the waiting time for the creation of one path the probability to get all paths is also high enough. l completion of stage k at time $\bar{T}_k$ we complete stage $k+1$ sufficiently fast. Assuming that at time $\bar{T}_k$ the population r^k-approximates the Pareto set $\mathbb{P}_{d(k)}$ with $d(k) = (3/2)^k$ it is possible to get an r^{k+1}-approximating path of the Pareto set $\mathbb{P}_{d(k+1)}$ with $d(k+1) = 3/2 \cdot d(k)$ into the population with sufficiently good probability. Note, that we only keep paths in the population that stem from non-dominated hyperboxes. Consequently, we restrict to the creation of those paths only fall within non-dominated hyperboxes. The precise length the paths within our population is not important, we only guarantee the approximation factor. Within the waiting time for the creation of one path the probability to get all paths is also high enough.

Lemma 4.43. *Assume that the population contains an r^i-approximating path for every Pareto-optimal path from the set $\mathbb{P}_d$, then the following holds: For every $d \in \mathbb{R}^+$ with $d \geq 1$ and every $\hat{P}_{(u,v)} \in \mathbb{P}_{(3/2)d} \setminus \mathbb{P}_d$, there exist at least $d/4$ different combinations $x \in V$ such that $P_{(u,x)}$ and $P_{(x,v)}$ are in the population and their concatenation at the vertex x results in a u-v-path $P_{u,v}$ which is a an r^{i+1}-approximation of $\hat{P}_{(u,v)}$.*

Proof. Let $\hat{P}_{(u,v)} = (u = u_0, u_1, u_2, \ldots, u_{\ell-1}, v = u_\ell)$ with $d < \ell \leq (3/2)d$ be in $\mathbb{P}_{(3/2)d}$ but not in the set $\mathbb{P}_d$. Consider the index set $J := \{\lceil d/2 \rceil, \ldots, \lfloor d \rfloor\}$, and let $j \in J$ and $x := u_j$ be a vertex of $\hat{P}_{(u,v)}$. As $\lfloor d \rfloor \leq d < \ell$ vertex x is well-defined. If we split up $\hat{P}_{(u,v)}$ in the two paths $\hat{P}_{(u,x)}$ and $\hat{P}_{(x,v)}$,

$$w(\hat{P}_{(u,v)}) = w(\hat{P}_{(u,x)}) + w(\hat{P}_{(x,v)})$$

then it is easy to see that those two paths are also Pareto-optimal, otherwise we could replace a subpath of $\hat{P}_{(u,v)}$ by one with less weight contradicting the Pareto-optimality of $\hat{P}_{(u,v)}$. Moreover, those two paths are from the set $\mathbb{P}_d$ and we have an r^i-approximating path $P_{(u,x)}$ and $P_{(x,v)}$ in our population by the assumption of the lemma. Concatenating $P_{(u,x)}$ and $P_{(x,v)}$ at their common

vertex x results in path $P_{(u,v)}$ which is an r^i-approximation of $\hat{P}_{(u,v)}$

$$\begin{aligned} w(P_{(u,v)}) &= w(P_{(u,x)}) + w(P_{(x,v)}) \\ &\leq r^i w(\hat{P}_{(u,x)}) + r^i \cdot w(\hat{P}_{(x,v)}) \\ &\leq r^i (w(\hat{P}_{(u,x)}) + w(\hat{P}_{(x,v)})) \\ &= r^i (w(\hat{P}_{(u,v)})). \end{aligned}$$

Due to the environmental selection present in DEMO (cf. Lemma 4.40) we might loose an additional factor r. The algorithm may replace the path $P_{(u,v)}$ by another one which is an r^{i+1}-approximation of $\hat{P}_{(u,v)}$.

Note, that we make no assumption about the lengths of the paths in the population.

Finally, the size of set J determines the number of choices we have for x. Since

$$|J| = \lfloor d \rfloor - \left\lceil \frac{d}{2} \right\rceil + 1 \geq \begin{cases} 1 - 1 + 1 \geq \frac{d}{4} & \text{if } 1 \leq d < 2, \\ 2 - 2 + 1 \geq \frac{d}{4} & \text{if } 2 \leq d < 4, \\ d - 1 - \frac{d}{2} - 1 + 1 \geq \frac{d}{4} & \text{if } d \geq 4, \end{cases}$$

there are at least $d/4$ ways to choose x. □

Applying the lemma to DEMO with feasible parent selection we get a lower bound of ${\mathcal{P}_{\max}}^{-1} n^{-1} {p_c}^{(3/2)^k}/4$ on the success probability to create for a path from the set $\mathbb{P}_{3/2d(k-1)}$ an r^k-approximating path in our population. That is, in stage k we set the variables $d := d(k-1)$ and $i := k-1$.

We will now prove Proposition 4.42 using Lemma 4.43.

Proof of Proposition 4.42. For $k \in \{1, \ldots, k^*\}$ with $k^* := 2 \log n$ from Equation (4.32) we show that Inequality (4.34) holds. At the end of stage $k-1$at time $\bar{T}_{k-1}$ we have for every Pareto-optimal path from the set $\mathbb{P}_{k-1}$ a path in our population which is an r^{k-1}-approximation. How long does it take to create get an r^k-approximating path into the population for every path in the set $\mathbb{P}_k$? Since the events under consideration, to get for every path from $\mathbb{P}_{(3/2)^k}$ an r^k-approximating representative into our population, do not have to be independent of each other we use Lemma 4.12 (Coupon Collector with Dependencies) to derive a bound on the probability for Inequality (4.34).

However, here we cannot plug in the set $\mathbb{P}_{k-1}$ and $\mathbb{P}_k$ as we might have expected from the previous proofs for the single-criteria problem. Consider the set of hyperboxes $\mathbb{B}_k := b_r(w(\mathbb{P}_k \setminus \mathbb{P}_{k-1}))$ with a size of at most $|\mathbb{B}_k| \leq \mathcal{P}_{\max}$. Choose for each non-dominated $B \in \mathbb{B}_k$ and for every pair of vertices $(u, v) \in V^2$ a path $\hat{P}_{u,v}$ that is mapped to this hyperbox, i. e. $b_r(w(\hat{P}_{u,v})) = B$. We

only consider non-dominated hyperboxes due to the environmental selection of DEMO (Line 10). Upon insertion of a non-dominated path into the population all paths from dominated hyperboxes are removed.

We set $I := \mathbb{B}_k \setminus \mathbb{B}_{k-1}$ and denote by $A^t_{u,v}$ the event that at time $\bar{T}_{k-1} + 1 + t$ for a non-dominated $B \in \mathbb{B}_k$ to which a path $\hat{P}_{u,v}$ is mapped we have an r^k-approximating path $P_{u,v}$ in our population. We show that there exists a $p \in (0,1)$ such that

$$\mathbf{Pr}\left[A^t_{(u,v)} \,\middle|\, \bigcap_{0 \le s < t} \overline{A^s_{(u,v)}}\right] \ge p \tag{4.35}$$

holds for all $t \in \mathbb{N}$ and all hyperboxes $B \in I$.

Due to the definition of I there is no r^{k-1}-approximating path in the population for a non-dominated hyperbox $B \in \mathbb{B}_{k-1}$ at time $\bar{T}_{k-1} + t$ with positive probability, such that again the conditional probability is well-defined.

Since, there exists for every non-dominated $B \in \mathbb{B}_{k-1}$ to which a path $\hat{P}_{u,v}$ is mapped a r^{k-1}-approximating path $P_{u,v}$ in our population, we are able to use Lemma 4.43. We apply Lemma 4.43 to a path from $\hat{P}_{u,v} \in I$ for $d = d(k-1)$ and $i = k-1$ to derive a probability of

$$p = \frac{d(k-1)}{4} \cdot \frac{p_c}{\mathcal{P}_{\max} \cdot n} = \frac{(3/2)^k}{6} \cdot \frac{p_c}{n\mathcal{P}_{\max}}$$

to perform crossover in DEMO and successfully concatenate the particular pair of paths from the population to a path $P_{u,v}$ which r^k-approximates $\hat{P}_{u,v}$. If at time $\bar{T}_{k-1} + t + 1$ one of the events $A^0_{(u,v)}, \dots A^t_{(u,v)}$ has occurred for every $B \in I$ this implies the event $\bar{T}_k \le \bar{T}_{k-1} + 1 + t$ we derive from Lemma 4.12 with $r := C_1(2/3)^k n\mathcal{P}_{\max} \log n$ and $C_1 := 30/p_c$ that

$$p \cdot r = \frac{(3/2)^k}{6} \cdot \frac{p_c}{n\mathcal{P}_{\max}} \cdot \frac{30(2/3)^k n\mathcal{P}_{\max} \log n}{p_c} = 5 \log n \ge 5 \ln(n).$$

And in order to achieve an $(1+\varepsilon)$-approximation the size of the hyperboxes r are scaled according to the number of stages $2 \log n$ (cf. Equation (4.32))

$$r = (1+\varepsilon)^{1/2 \log n}. \tag{4.36}$$

Therefore since $|I| \le \mathcal{P}_{\max} \le n^2 \cdot (2 \log n)^{d-1} \left(\frac{\log(n \cdot w_{\max})}{\log(1+\varepsilon)}\right)^{d-1}$ (cf. Equation (4.30))

$$\begin{aligned} \mathbf{Pr}\left[\bar{T} \le \bar{T}_{k-1} + 1 + r\right] &\le |I| \cdot \mathrm{e}^{-5 \ln(n)} \\ &\le \frac{1}{n^2}. \end{aligned}$$

This completes the proof of our main theorem. □

4.8 Discussion

In the following, we want to discuss the insights that can be gained from the analyses for APSP. We start by discussing the problem characteristics that were used in our analysis. The analyses for APSP make use of the fact that a shortest path between two vertices u and v can be obtained by combining two subpaths. In this way crossover between two individuals can lead to a shortest path between u and v. Our runtime bounds can be extended to other but linear weight functions. For this, notice that the only place in the proofs of our runtime bounds where we refer to the actual properties of the weight-function is Lemma 4.9. In other words, if a fitness function f on all paths of a graph G satisfies Lemma 4.9, then our upper runtime bounds also hold for f. This motivates the following definition.

> **Definition 4.44.** *Let G be a finite, strongly connected directed graph and let f be a non-negative fitness function on the set of all paths in G. Then f is called* subpath optimal *if the following holds.*
>
> *If $P_{u,v} = (u = u_0, \dots, u_\ell = v)$ is an f-optimal u-v-path of length ℓ and $x = u_i$ and $y = u_j$ are vertices of $P_{u,v}$ with $0 \leq i \leq j \leq \ell$, then substituting the f-optimal subpath $P_{x,y}$ with another f-optimal path $P'_{x,y}$ yields an f-optimal path $P'_{u,v}$.*

Substituting a subpath $P_{x,y}$ of $P_{u,v}$ amounts to the concatenation of the three paths $P_{u,x}$ with $P'_{x,y}$ and $P_{y,v}$ at the vertices x and y (where one path can possibly be empty). Recall, that by design our crossover operators used in the Steady State GA_{APSP} concatenate two or three paths to derive $P_{u,v}$. As in our analyses we ignored the ability of mutation to shrink paths by deleting edges, we regard the application of mutation as a concatenation of two paths.

We can now apply Lemma 4.9 to any weight-function with non-negative weights that is *subpath optimal*. This worked for the subpath optimal fitness function associated with APSP that maps paths to their lengths. Another subpath optimal example is mapping paths to the weight of their lightest edge (and maximize); this is known as the *all-pairs bottleneck paths* problem (see [52, 154]) and has applications for example in voting theory [140]. Lemma 4.9 is also applicable because if the minimum increases on a subpath then the overall minimum of the path is never decreased.

Thus, as a direct corollary to Theorem 4.5, the Steady State GA_{APSP} with repair on the all-pairs bottleneck paths problem has an optimization time

of $O\big(n^{3.2}(\log n)^{0.2}\big)$ iterations with high probability; and as a corollary to Theorem 4.6, we have that the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ with feasible parent selection on the all-pairs bottleneck paths problem has an optimization time of $O\big(n^3 \log n\big)$ iterations with high probability.

Relation to dynamic programming As a final remark we would like to relate the success of our GA to dynamic programming. Our analysis is oriented at the Bellman-Ford Algorithm; the stages we consider are the same stages that occur there. From this perspective, one may say that the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ mimics the optimization behavior of the Bellman-Ford Algorithm.

The used representation of the individuals and the particular diversity mechanism is still natural since we would like to compute shortest paths for each pair of vertices. We see that the randomness introduced by the selection and crossover operators in the Steady State $\mathrm{GA}_{\mathrm{APSP}}$ raises subtle points in the analysis of our runtime bounds which are not present in the analysis of the Bellman-Ford Algorithm. Two examples are the necessary overlap of the optimal subpaths in crossover and, in the case of Crossover with Repair, the interplay between mutation and crossover.

In a more general setting, one may ask whether our findings relate to other problems that have dynamic programming algorithms. The answer to this again relates to Lemma 4.9 and Definition 4.44. If the problem structure and representation within a GA is such that with sufficiently large probability the crossover operator can combine two optimal solutions like in Definition 4.44, we may hope that a GA using crossover and mutation outperforms a GA using mutation only. However, we are not aware of any concrete examples of such optimization problems, except for APSP with subpath optimal fitness functions.

5 Discussion

We outline the results achieved and highlight connections and further insights. In the introduction we discussed that it needs some careful investigation in which settings a specific evolutionary algorithm (EA) performs well. For EAs and problems with a dynamic programming formulation we have examined in Chapter 2 how to choose a representation such that the EA obtains the ability to carry out dynamic programming (DP). Based on a general framework for dynamic programming, we have given a framework for evolutionary algorithms that have a dynamic programming ability. We analyzed the optimization time of such an algorithm depending on the corresponding dynamic programming approach. By considering some well-known combinatorial optimization problems, we have shown that our framework captures all known DP-based evolutionary algorithms and allows to treat other related problems. We carried out supplementary experiments using a $(\mu+1)$ EA and GA for the traveling salesperson problem (TSP) based on its dynamic programming formulations. We used a variety of repair crossover operators and evaluated their optimization time experimentally. Although the optimization time decreases for the $(\mu+1)$ GA this is not as significant as one might hope for. A DP formulation usually involves the creation of a large number of states and hence we need to store a large number of individuals in our population. This certainly prevents our GA from becoming much faster than the comparable EA.

Successfully using crossover to speed up the optimization time needs a careful exploitation of its working principles (compare page 11). Crossover is a very useful operator if the population is sufficiently diverse. Our considerations in Chapter 3 of the shuffle GA have revealed the potential of uniform crossover in populations with good diversity. The bit shuffling operator simulates individuals that have evolved independently on a function of unitation. In this setting crossover leads to surprising and drastic speedups from $\Theta(n \log n)$ function evaluations to only $\Theta(\sqrt{n})$ for OneMax. For Jump_k the gap is even

larger, namely $\Theta(n^k)$ versus $\Theta(\sqrt{n} + 4^k)$. In a more realistic setting of the (μ+1) GA we have shown that also small populations can develop sufficient diversity for $\textsc{Jump}_k$ if the crossover probability is small enough. If it is set too large, crossover makes the population collapse to a single individual, leading to superpolynomial running times on $\textsc{Jump}_k$. Together with our Monte Carlo simulations we have provided a deepened insight into the opposing effects that mutation and crossover have on the convex hull and thus the diversity of the population.

Considering the problems that come along with understanding the dynamics in a population it was a major breakthrough that Doerr et al. [44] could rigorously analyze a genetic algorithm for the all-pairs shortest path problem (APSP), which still is the only non-artificial problem for which the usefulness of crossover could be proven. In this thesis, in Chapter 4, we showed for the single-criteria that, unlike the previous analysis suggests, for most of the time during a typical run of the algorithm, both mutation and crossover contribute significantly to the progress. Crossover makes large jumps in the search space and mutation explores the closer neighborhood. This improved understanding of how crossover and mutation interact, in particular, that they do interact at all, gives rise to an improved guarantee for the optimization time of the genetic algorithm. We showed that with high probability, $O(n^{3.25}(\log n)^{1/4})$ iterations suffice to find shortest paths between any two vertices in an n–vertex graph. This is asymptotically optimal. Using a suitably chosen class of graphs, we showed that for a sufficiently small constant $C > 0$, with high probability after $Cn^{3.25}(\log n)^{1/4}$ iterations not all shortest paths are found. While the proof of this result looks simple, it is remarkable that such a bound could be shown, (i) due to the unusual runtime including a $\log^{1/4}(n)$ factor, (ii) because the good interplay of mutation and crossover makes it hard to prove lower bounds for optimization times, and (iii) because the result marks a quite strict separation between success and failure—already reducing the time by a constant factor changes the behavior from "works with high probability" to "fails with high probability". Compared to the $\Theta(n^4)$ for an evolutionary algorithm, our genetic algorithm is surprisingly fast.

Additionally, in Chapter 4 we studied extensions of the basic algorithm using repair mechanisms. We showed that repairing infeasible offspring leads to an improved optimization time of $O(n^{3.2}(\log n)^{1/5})$. As a second part of our study we proved that choosing parents such that only feasible offspring are created guarantees an even better optimization time of $O(n^3 \log n)$. Both results show that already simple recombination strategies can asymptotically improve the optimization time of evolutionary algorithms. Apart from the rigorously proven results we conjecture that the bounds are actually tight and cannot be improved by a better analysis of the process. In the remainder

of Chapter 4 we extended this approach to the NP-hard multi-criteria APSP using a specific diversity mechanism to control the size of the population. We conducted a rigorous runtime analysis and proved that our algorithm is a fully-polynomial time randomized approximation scheme (FPRAS). This was the first time that rigorous runtime analyses have shown the usefulness of crossover for an NP-hard optimization problem. We finally discussed that the successful application and analysis of genetic algorithm for APSP carries over to other problems that possess a specific structure. We also gave some reasons why it is not easily possible to generalize those insights to the large class of problems with a DP structure connecting the experiments of Chapter 2 with our DP-problem treated in Chapter 4.

In summary, with our work we improved the understanding of evolutionary algorithms on a large class of combinatorial optimization problem. We gave new insight in the working principles of crossover, proposed improved methods to analyze algorithms, and finally proved upper and in some cases also lower bounds for the optimization time of genetic algorithms.

5.1 Future Work

In the following, we point out some topics for future research.

Our experiments for TSP empirically show that it is possible to improve the optimization time of the generic evolutionary algorithm (Algorithm 2.2) for dynamic programming problems in Section 2.4. This intuition is further supported by our improved polynomial bounds on the optimization time of a genetic algorithm for APSP, which also has a dynamic programming structure. It is an open question whether it is possible to find another DP-problem for which we can mathematically prove the usefulness of crossover. We refer the reader back to Section 4.8 where subtle points involved with this question are discussed.

Another question related to the one just raised is whether there is a combinatorial optimization problem whose optimization time drops from superpolynomial to polynomial if a GA is used instead of an EA. This would certainly be a milestone in the understanding of crossover in evolutionary computation.

The genetic algorithm examined for APSP makes use of a strong diversity mechanism (each individual represents a path between a different pair of vertices). Often evolutionary algorithms use much weaker diversity mechanism such as niching, deterministic crowding and fitness sharing and the goal is to

compute just a single solution instead of a set of solutions. We state it as an open problem to show that crossover speeds up evolutionary algorithms for single-objective combinatorial optimization using one of the stated diversity mechanisms. On the other hand, multi-objective problems use in a natural way a diversity mechanisms according to Pareto dominance. Often the population of an evolutionary algorithm for multi-objective optimization contains a population which represents the different tradeoffs with respect to the given objective function at a certain time step. It would be interesting to have rigorous results that show the usefulness of crossover in evolutionary multi-objective optimization.

In the discussion of mixability in the introduction we state that Livnat et al. [98] argue that evolution is "survival of the most mixable" in contrast to the Darwinian hypothesis of the "survival of the fittest". Intuitively, this means that sexual recombination in biology serves to increase the "genetic robustness" [5]. However, in the literature on genetic algorithms this thought has not been tackled so far. It would be an interesting approach to investigate whether crossover is able to create some notion of robustness in dynamically changing environments.

But in order to achieve this progress the foundation for the research on crossover needs to be improved first. There is still a significant lack in the understanding of the dynamics in populations. These issues need to be solved before further results for combinatorial optimization problems can be achieved.

Bibliography

All references coauthored during the Ph.D. Studies, which are cited within this thesis, are marked with "•". For each publication coauthored with n authors my contribution is at least $1/n$. Additionally, all entries are crossreferenced with the pages where they are cited.

[1] Y. Afek, N. Alon, O. Barad, E. Hornstein, N. Barkai, and Z. Bar-Joseph. A biological solution to a fundamental distributed computing problem. *Science*, 331:183–185, 2011. (page 3).

[2] N. Alon and J. H. Spencer. *The Probabilistic Method*. Wiley, 3rd edition, 2008. (pages 76, 78).

[3] S. Arora, Y. Rabani, and U. V. Vazirani. Simulating quadratic dynamical systems is pspace-complete. In *Proceedings of the 26th Symposium on Theory of Computing Conference (STOC)*, pp. 459–467. ACM Press, 1994. (page 2).

[4] A. Auger and B. Doerr, editors. *Theory of Randomized Search Heuristics: Foundations and Recent Developments*. World Scientific, 2011. (pages 2, 7).

[5] R. B. R. Azevedo, R. Lohaus, S. Srinivasan, K. K. Dang, and C. L. Burch. Sexual reproduction selects for robustness and negative epistasis in artificial gene networks. *Nature*, 440:87–90, 2006. (page 121).

[6] N. H. Barton and B. Charlesworth. Why sex and recombination? *Science*, 281:1986–1990, 1998. (page 8).

[7] S. Baswana, S. Biswas, B. Doerr, T. Friedrich, P. P. Kurur, and F. Neumann. Computing single source shortest paths using single-objective fitness functions. In *Proceedings of the 10th International Workshop on Foundations of Genetic Algorithms (FOGA)*, pp. 59–66. ACM Press, 2009. (page 64).

[8] R. E. Bellman and S. E. Dreyfus. *Applied Dynamic Programming*. Princeton University Press, 1962. (pages 15, 21).

[9] N. Beume, C. M. Fonseca, M. López-Ibáñez, L. Paquete, and J. Vahrenhold. On the complexity of computing the hypervolume indicator.

IEEE Transactions on Evolutionary Computation, 13:1075–1082, 2009. (page 7).

[10] L. Bianchi, M. Dorigo, L. M. Gambardella, and W. J. Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8:239–287, 2009. (page 2).

[11] C. Blum, R. Chiong, M. Clerc, K. A. D. Jong, Z. Michalewicz, F. Neri, T. Weise, and T. Weise. Evolutionary optimization. In *Variants of Evolutionary Algorithms for Real-World Applications*, pp. 1–29. Springer-Verlag, 2012. (page 3).

[12] V. Bonifaci, K. Mehlhorn, and G. Varma. Physarum can compute shortest paths. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 233–240, 2012. (page 3).

[13] K. Bringmann and T. Friedrich. Parameterized average-case complexity of the hypervolume indicator. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*. ACM Press, 2013. (page 7).

[14] K. Bringmann and T. Friedrich. Approximation quality of the hypervolume indicator. *Artificial Intelligence*, 195:265–290, 2013. (page 7).

[15] K. Bringmann, T. Friedrich, F. Neumann, and M. Wagner. Approximation-guided evolutionary multi-objective optimization. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1198–1203, 2011. (page 104).

[16] D. Brockhoff, T. Friedrich, and F. Neumann. Analyzing hypervolume indicator based algorithms. In *Proceedings of the 10th International Conference Parallel Problem Solving From Nature (PPSN)*, Vol. 5199 of *Lecture Notes in Computer Science*, pp. 651–660. Springer-Verlag, 2008. (page 7).

[17] D. Brockhoff, T. Friedrich, N. Hebbinghaus, C. Klein, F. Neumann, and E. Zitzler. On the effects of adding objectives to plateau functions. *IEEE Transactions on Evolutionary Computation*, 13:591–603, 2009. (page 7).

[18] B. Chazelle. Natural algorithms. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 422–431. SIAM, 2009. (page 3).

[19] B. Chazelle. Natural algorithms and influence systems. *Communications of the ACM*, 55:101–110, 2012. (page 3).

[20] T. Chen, J. He, G. Sun, G. Chen, and X. Yao. A new approach for analyzing average time complexity of population-based evolutionary algorithms on unimodal problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 39:1092–1106, 2009. (page 12).

[21] D. M. Cherba and W. F. Punch. Crossover gene selection by spatial location. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1111–1116. ACM Press, 2006. (page 68).

[22] R. Chowdhury, L. Hai-Son, and V. Ramachandran. Cache-oblivious dynamic programming for bioinformatics. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7:495–510, 2010. (page 18).

[23] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001. (pages 18, 28).

[24] J. Culberson. Genetic invariance: A new paradigm for genetic algorithm design. Technical Report Tech. Rep. TR92-02, Univ. Alberta, Edmonton AB, Canada, 1992. (page 44).

[25] C. Daskalakis, I. Diakonikolas, and M. Yannakakis. How good is the Chord algorithm? In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 978–991, 2010. (page 103).

[26] I. Diakonikolas and M. Yannakakis. Small approximate Pareto sets for biobjective shortest paths and other problems. *SIAM Journal on Computing*, 39:1340–1371, 2009. (page 103).

[27] M. Dietzfelbinger, B. Naudts, C. Van Hoyweghen, and I. Wegener. The analysis of a recombinative hill-climber on H-IFF. *IEEE Transactions on Evolutionary Computation*, 7:417–423, 2003. (page 44).

[28] Y. Disser, M. Müller-Hannemann, and M. Schnee. Multi-criteria shortest paths in time-dependent train networks. In *Proceedings of the 7th International Conference on Experimental Algorithms (WEA)*, pp. 347–361, Berlin, Heidelberg, 2008. Springer-Verlag. (page 100).

[29] B. Doerr and L. A. Goldberg. Adaptive drift analysis. *Algorithmica*, 65: 224–250, 2013. (page 12).

[30] B. Doerr and D. Johannsen. Edge-based representation beats vertex-based representation in shortest path problems. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 759–766. ACM Press, 2010. (page 65).

[31] B. Doerr and D. Johannsen. Adjacency list matchings – an ideal genotype for cycle covers. In *Proceedings of the Genetic and Evolutionary*

Computation Conference (GECCO), pp. 1203–1210. ACM Press, 2007. (page 29).

• [32] B. Doerr and M. Theile. Improved analysis methods for crossover-based algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 247–254. ACM Press, 2009. (pages 16, 60, 63, 64, 68, 69, 70, 74, 75, 85, 93, 94).

[33] B. Doerr, N. Hebbinghaus, and F. Neumann. Speeding up evolutionary algorithms by restricted mutation operators. In *Proceedings of the 9th International Conference Parallel Problem Solving From Nature (PPSN)*, Vol. 4193 of *Lecture Notes in Computer Science*, pp. 978–987, Berlin, Germany, 2006. Springer-Verlag. (page 6).

[34] B. Doerr, E. Happ, and C. Klein. A tight bound for the (1+1) EA on the single source shortest path problem. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pp. 1890–1895, Singapore, 2007. IEEE Press. (pages 63, 64, 65).

[35] B. Doerr, N. Hebbinghaus, and F. Neumann. Speeding up evolutionary algorithms through asymmetric mutation operators. *Evolutionary Computation*, 15:401–410, 2007. (pages 6, 29).

[36] B. Doerr, C. Klein, and T. Storch. Faster evolutionary algorithms by superior graph representations. In *Proceedings of the 1st IEEE Symposium on Foundations of Computational Intelligence (FOCI)*, pp. 245–250. IEEE Press, 2007. (pages 7, 29).

[37] B. Doerr, E. Happ, and C. Klein. Crossover can provably be useful in evolutionary computation. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 539–546. ACM Press, 2008. (pages 10, 63, 67, 68, 69).

• [38] B. Doerr, A. V. Eremeev, C. Horoba, F. Neumann, and M. Theile. Evolutionary algorithms and dynamic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 771–778. ACM Press, 2009. (pages 17, 18).

• [39] B. Doerr, D. Johannsen, T. Kötzing, F. Neumann, and M. Theile. More effective crossover operators for the all-pairs shortest path problem. In *Proceedings of the 11th International Conference Parallel Problem Solving From Nature (PPSN)*, Vol. 6238 of *Lecture Notes in Computer Science*, pp. 184–193. Springer-Verlag, 2010. (pages 60, 63, 69).

• [40] B. Doerr, A. V. Eremeev, F. Neumann, M. Theile, and C. Thyssen. Evolutionary algorithms and dynamic programming. *Theoretical Computer Science*, 412:6020–6035, 2011. (pages 17, 18).

[41] B. Doerr, M. Fouz, and C. Witt. Sharp bounds by probability-generating functions and variable drift. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 2083–2090, 2011. (page 12).

[42] B. Doerr, E. Happ, and C. Klein. Tight analysis of the (1+1) EA for the single source shortest path problem. *Evolutionary Computation*, 19: 673–691, 2011. (pages 63, 64, 65).

[43] B. Doerr, D. Johannsen, T. Kötzing, P. K. Lehre, M. Wagner, and C. Winzen. Faster black-box algorithms through higher arity operators. In *Proceedings of the 11th International Workshop on Foundations of Genetic Algorithms (FOGA)*, pp. 163–172. ACM, 2011. (page 11).

[44] B. Doerr, E. Happ, and C. Klein. Crossover can provably be useful in evolutionary computation. *Theoretical Computer Science*, 425:17–33, 2012. (pages 10, 16, 25, 36, 60, 63, 64, 65, 67, 71, 87, 89, 95, 119).

[45] B. Doerr, D. Johannsen, and C. Winzen. Multiplicative drift analysis. *Algorithmica*, 64:673–697, 2012. (page 12).

• [46] B. Doerr, D. Johannsen, T. Kötzing, F. Neumann, and M. Theile. More effective crossover operators for the all-pairs shortest path problem. *Theoretical Computer Science*, 471:12–26, 2013. (pages 60, 63, 64, 69, 75).

[47] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, 2004. (page 5).

[48] S. Droste. Analysis of the (1+1) EA for a dynamically bitwise changing OneMax. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 909–921. Springer, 2003. (page 6).

[49] S. Droste, T. Jansen, and I. Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276:51–81, 2002. (pages 6, 14, 44).

[50] S. Droste, T. Jansen, and I. Wegener. Optimization with randomized search heuristics – The (A)NFL theorem, realistic scenarios, and difficult functions. *Theoretical Computer Science*, 287:131–144, 2002. (page 6).

[51] S. Droste, T. Jansen, and I. Wegener. Upper and lower bounds for randomized search heuristics in black-box optimization. *Theory of Computing Systems*, 39:525–544, 2006. (pages 44, 48).

[52] R. Duan and S. Pettie. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 384–391. SIAM, 2009. (page 116).

[53] F. Duddeck. Multidisciplinary optimization of car bodies. *Structural and Multidisciplinary Optimization*, 35:375–389, 2008. (page 2).

[54] M. Ehrgott. *Multicriteria optimization.* Springer-Verlag, 2nd edition, 2005. (page 102).

[55] A. V. Eremeev. On complexity of optimal recombination for binary representations of solutions. *Evolutionary Computation*, 16:127–147, 2008. (page 16).

[56] A. V. Eremeev. Some fully polynomial time randomized approximation scheme based on an evolutionary algorithm. *Journal of Applied and Industrial Mathematics*, 5:322–330, 2011. Translation of Russian article (А. В. Еремеев. Вполне полиномиальная рандомизированная аппроксимационная схема на основе эволюционного алгоритма. *дискретный анализ и исследование операций*, 17:3–17, 2010). (page 17).

[57] Y. G. Evtushenko and M. Potapov. Methods of numerical solution of multicriterion problem. In *Soviet mathematics – doklady*, Vol. 34, pp. 420–423, 1987. (page 103).

[58] W. Feller. *An introduction to probability theory and its applications. Vol. I.* Third edition. John Wiley & Sons Inc., New York, 1968. (pages 12, 72).

[59] M. R. Fellows and R. G. Downey. *Parameterized Complexity.* Springer Verlag, 1998. (page 7).

[60] S. Fischer and I. Wegener. The one-dimensional Ising model: Mutation versus recombination. *Theoretical Computer Science*, 344:208–225, 2005. (page 10).

[61] R. A. Fisher. The correlation between relatives on the supposition of Mendelian inheritance. *Transactions of the Royal Society of Edinburgh*, 52:399–433, 1918. (page 9).

[62] J. Flum and M. Grohe. *Parameterized Complexity Theory.* Springer Verlag, 2006. (page 7).

[63] T. Friedrich, N. Hebbinghaus, and F. Neumann. Rigorous analyses of simple diversity mechanisms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1219–1225. ACM Press, 2007. (page 6).

[64] T. Friedrich, N. Hebbinghaus, F. Neumann, J. He, and C. Witt. Approximating covering problems by randomized search heuristics using

multi-objective models. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 797–804. ACM Press, 2007. (page 7).

[65] T. Friedrich, P. S. Oliveto, D. Sudholt, and C. Witt. Analysis of diversity-preserving mechanisms for global exploration. *Evolutionary Computation*, 17:455–476, 2009. (pages 6, 43).

[66] T. Friedrich, J. He, N. Hebbinghaus, F. Neumann, and C. Witt. Approximating covering problems by randomized search heuristics using multi-objective models. *Evolutionary Computation*, 18:617–633, 2010. (page 7).

[67] M. Garey and D. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, San Francisco, 1979. (page 61).

[68] O. Giel and I. Wegener. Evolutionary algorithms and the maximum matching problem. In *Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, Vol. 2607 of *Lecture Notes in Computer Science*, pp. 415–426, Berlin, Germany, 2003. Springer-Verlag. (page 7).

[69] P. Hansen. Bicriterion path problems. In *Multiple Criteria Decision Making: Theory and Applications*, Vol. 177 of *Lecture Notes in Economics and Mathematical Systems*, pp. 109–127. Springer-Verlag, 1980. (page 103).

[70] C. N. Harper and L. Davis. Evolutionary computation at American Air Liquide. In *Evolutionary Computation in Practice*, Vol. 88 of *Studies in Computational Intelligence*, pp. 313–317. Springer-Verlag, 2008. (page 2).

[71] J. He and X. Yao. A study of drift analysis for estimating computation time of evolutionary algorithms. *Natural Computing*, 3:21–35, 2004. (page 48).

[72] M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10:196–210, 1962. (pages 25, 26).

[73] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975. (page 8).

[74] R. Horner. *A taxonomic guide to some common marine phytoplankton.* Biopress Ltd, 2002. (page 8).

[75] C. Horoba. Analysis of a simple evolutionary algorithm for the multiobjective shortest path problem. In *Proceedings of the 10th International Workshop on Foundations of Genetic Algorithms (FOGA)*, pp. 113–120. ACM Press, 2009. (pages 7, 16, 69, 103, 110).

[76] C. Horoba and F. Neumann. Benefits and drawbacks for the use of epsilon-dominance in evolutionary multi-objective optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 641–648. ACM Press, 2008. (pages 7, 104).

[77] P. Horrocks, R. Pinches, Z. Christodoulou, S. Kyes, and C. Newbold. Variable var transition rates underlie antigenic variation in malaria. *Proceedings of the National Academy of Sciences, USA*, 101:11129–11134, 2004. (page 31).

[78] M. Ibrahimov, A. Mohais, and Z. Michalewicz. Global optimization in supply chain operations. In *Natural Intelligence for Scheduling, Planning and Packing Problems*, pp. 1–28. Springer-Verlag, 2009. (page 2).

[79] J. Jägersküpper and T. Storch. How comma selection helps with the escape from local optima. In *Proceedings of the 9th International Conference Parallel Problem Solving From Nature (PPSN)*, Vol. 4193 of *Lecture Notes in Computer Science*, pp. 52–61. Springer-Verlag, 2006. (page 6).

[80] J. Jägersküpper and T. Storch. When the plus strategy outperforms the comma strategy and when not. In *Proceedings of the 1st IEEE Symposium on Foundations of Computational Intelligence (FOCI)*, pp. 25–32. IEEE Press, 2007. (page 6).

[81] T. Jansen. *Analyzing Evolutionary Algorithms – The Computer Science Perspective*. Springer-Verlag, 2013. (page 2).

[82] T. Jansen and U. Schellbach. Theoretical analysis of a mutation-based evolutionary algorithm for a tracking problem in the lattice. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 841–848. ACM Press, 2005. (page 6).

[83] T. Jansen and D. Sudholt. Design and analysis of an asymmetric mutation operator. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, IEEE Press, pp. 497–504, 2005. (page 6).

[84] T. Jansen and I. Wegener. Evolutionary algorithms – how to cope with plateaus of constant fitness and when to reject strings of the same fitness. *IEEE Transactions on Evolutionary Computation*, 5:589–599, 2001. (page 6).

[85] T. Jansen and I. Wegener. Real royal road functions – where crossover provably is essential. *Discrete Applied Mathematics*, 149:111–125, 2005. (pages 10, 11).

[86] T. Jansen and I. Wegener. The analysis of evolutionary algorithms – a proof that crossover really can help. *Algorithmica*, 34:47–66, 2002. (pages 10, 43, 48, 49, 50, 51, 56).

[87] J. Kennedy, R. C. Eberhart, and Y. Shi. *Swarm Intelligence*. Morgan Kaufmann, 2001. (page 5).

[88] R. Klötzler. Multiobjective dynamic programming. *Mathematische Operationsforschung und Statistik. Series Optimization*, 9:423–426, 1978. (pages 15, 21).

[89] J. D. Knowles and D. Corne. A comparison of encodings and algorithms for multiobjective spanning tree problems. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pp. 544–551. IEEE Press, 2001. (page 7).

• [90] T. Kötzing, D. Sudholt, and M. Theile. How crossover helps in pseudo-boolean optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 989–996. ACM Press, 2011. **Best Paper Award at GECCO 2011**. (page 42).

[91] S. Kratsch and F. Neumann. Fixed-parameter evolutionary algorithms and the vertex cover problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 293–300. ACM Press, 2009. (page 7).

[92] J. Lässig and D. Sudholt. The benefit of migration in parallel evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1105–1112. ACM Press, 2010. (page 7).

[93] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler. Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary Computation*, 10:263–282, 2003. (pages 7, 103, 105).

[94] M. Laumanns, L. Thiele, and E. Zitzler. Running time analysis of multiobjective evolutionary algorithms on pseudo-boolean functions. *IEEE Transactions on Evolutionary Computation*, 8:170–182, 2004. (page 7).

[95] P. K. Lehre. Fitness-levels for non-elitist populations. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 2075–2082. ACM Press, 2011. (pages 7, 12).

[96] P. K. Lehre. Negative drift in populations. In *Proceedings of the 11th International Conference Parallel Problem Solving From Nature (PPSN)*, pp. 244–253. Springer-Verlag, 2010. (pages 7, 12).

[97] P. K. Lehre and C. Witt. Black box search by unbiased variation. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1441–1448. ACM Press, 2010. (pages 44, 48).

[98] A. Livnat, C. Papadimitriou, J. Dushoff, and M. W. Feldmann. A mixability theory for the role of sex in evolution. *Proceedings of the National Academy of Sciences (PNAS)*, 105:19803–19808, 2008. (pages 3, 9, 121).

[99] A. Livnat, C. Papadimitriou, N. Pippenger, and M. W. Feldmann. Sex, mixability, and modularity. *Proceedings of the National Academy of Sciences (PNAS)*, 107:1452–1457, 2010. (page 3).

[100] P. Loridan. ε-solutions in vector minimization problems. *Journal of Optimization Theory and Applications*, 43:265–276, 1984. (page 103).

[101] K. Mehlhorn and P. Sanders. *Algorithms and Data Structures: The Basic Toolbox*. Springer-Verlag, 2008. (page 64).

[102] Z. Michalewicz and D. B. Fogel. *How to solve it: Modern heuristics*. Springer, Berlin, 2004. (pages 16, 68).

[103] Z. Michalewicz, G. Nazhiyath, and M. Michalewicz. A note on usefulness of geometrical crossover for numerical optimization problems. In *Proceedings of the Evolutionary Programming (EP)*, pp. 305–312. MIT Press, 1996. (page 68).

[104] Z. Michalewicz, M. Schmidt, M. Michalewicz, and C. Chiriac. Case study: An intelligent decision-support system. *IEEE Intelligent Systems*, 20:44–49, 2005. (page 2).

[105] Z. Michalewicz, M. Schmidt, M. Michalewicz, and C. Chiriac. Adaptive business intelligence: Three case studies. In *Evolutionary Computation in Dynamic and Uncertain Environments*, pp. 179–196. Springer-Verlag, 2007. (page 2).

[106] L. G. Mitten. Composition principles for synthesis of optimal multistage processes. *Operations Research*, 12:610–619, 1964. (page 21).

[107] A. Moraglio. Abstract convex evolutionary search. In *Proceedings of the 11th International Workshop on Foundations of Genetic Algorithms (FOGA)*, pp. 151–162. ACM Press, 2011. (pages 11, 43).

[108] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. (page 12).

[109] F. Neumann. Expected runtimes of evolutionary algorithms for the Eulerian cycle problem. *Computers and Operations Research*, 35:2750–2759, 2008. (pages 7, 29).

[110] F. Neumann and J. Reichel. Approximating minimum multicuts by evolutionary multi-objective algorithms. In *Proceedings of the 10th International Conference Parallel Problem Solving From Nature (PPSN)*, Vol. 5199 of *Lecture Notes in Computer Science*, pp. 72–81. Springer-Verlag, 2008. (page 7).

• [111] F. Neumann and M. Theile. How crossover speeds up evolutionary algorithms for the multi-criteria all-pairs-shortest-path problem. In *Proceedings of the 11th International Conference Parallel Problem Solving From Nature (PPSN)*, Vol. 6238 of *Lecture Notes in Computer Science*, pp. 667–676. Springer-Verlag, 2010. (pages 7, 16, 60, 64, 69, 100).

[112] F. Neumann and I. Wegener. Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. *Theoretical Computer Science*, 378:32–40, 2007. (page 7).

[113] F. Neumann and C. Witt. *Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity*. Springer-Verlag, 2010. (pages 2, 7).

[114] F. Neumann, J. Reichel, and M. Skutella. Computing minimum cuts by randomized search heuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 779–786. ACM Press, 2008. (page 7).

[115] F. Neumann, P. S. Oliveto, and C. Witt. Theoretical analysis of fitness-proportional selection: landscapes and efficiency. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 835–842. ACM Press, 2009. (page 6).

[116] A. Q. Nguyen, A. M. Sutton, and F. Neumann. Population size matters: Rigorous runtime results for maximizing the hypervolume indicator. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*. ACM Press, 2013. (page 7).

[117] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. (page 7).

[118] P. S. Oliveto and C. Witt. Simplified drift analysis for proving lower bounds in evolutionary computation. *Algorithmica*, 59:369–386, 2011. (page 12).

[119] P. S. Oliveto and C. Witt. Simplified drift analysis for proving lower bounds in evolutionary computation. In *Proceedings of the 10th International Conference Parallel Problem Solving From Nature (PPSN)*, Vol. 5199 of *Lecture Notes in Computer Science*, pp. 82–91. Springer-Verlag, 2008. (page 12).

[120] P. S. Oliveto, J. He, and X. Yao. Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results. *International Journal of Automation and Computing*, 4:281–293, 2007. (page 12).

[121] C. H. Papadimitriou and M. Yannakakis. Multiobjective query optimization. In *Proceedings of the 20th ACM Symposium on Principles of Database Systems (PODS)*, pp. 52–59. ACM Press, 2001. (page 103).

[122] C. H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *Proceedings of the 41st Symposium Foundations of Computer Science (FOCS)*, pp. 86–92. IEEE Computer Society, 2000. (page 103).

[123] C. Potts and M. Kovalyov. Scheduling with batching: A review. *European Journal of Operational Research*, 120:228–249, 2000. (page 18).

[124] Y. Rabani, Y. Rabinovich, and A. Sinclair. A computational view of population genetics. *Random Structures and Algorithms*, 12:313–334, 1998. (pages 2, 8).

[125] Y. Rabinovich and A. Wigderson. An analysis of a simple genetic algorithm. In *In Proceedings of the 4th International Conference on Genetic Algorithms*, pp. 215–221. Morgan Kaufmann, 1991. (page 8).

[126] Y. Rabinovich, A. Sinclair, and A. Wigderson. Quadratic dynamical systems. In *Proceedings of the 33rd Symposium Foundations of Computer Science (FOCS)*, pp. 304–313. IEEE Computer Society, 1992. (page 2).

[127] G. R. Raidl and B. A. Julstrom. Edge sets: an effective evolutionary coding of spanning trees. *IEEE Transactions on Evolutionary Computation*, 7:225–239, 2003. (pages 7, 16).

[128] J. Reichel and M. Skutella. Evolutionary algorithms and matroid optimization problems. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 947–954. ACM Press, 2007. (page 7).

[129] J. Reichel and M. Skutella. Evolutionary algorithms and matroid optimization problems. *Algorithmica*, 57:187–206, 2010. (page 15).

[130] H. Reuter. An approximation method for the efficiency set of multiobjective programming problems. *Optimization*, 21:905–911, 1990. (page 103).

[131] J. N. Richter, A. Wright, and J. Paxton. Ignoble trails – where crossover is provably harmful. In *Proceedings of the 10th International Conference Parallel Problem Solving From Nature (PPSN)*, Vol. 5199 of *Lecture Notes in Computer Science*, pp. 92–101. Springer-Verlag, 2008. (page 10).

[132] P. Rohlfshagen, P. K. Lehre, and X. Yao. Dynamic evolutionary optimisation: an analysis of frequency and magnitude of change. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1713–1720. ACM Press, 2009. (page 6).

[133] F. Rothlauf. *Representations for genetic and evolutionary algorithms*. Springer-Verlag, 2nd edition, 2006. (pages 7, 16).

[134] G. Rudolph. *Convergence Properties of Evolutionary Algorithms*. Kovač, 1997. (page 6).

[135] G. Ruhe and B. Fruhwirth. ε-optimality for bicriteria programs and its application to minimum cost flows. *Computing*, 44:21–34, 1990. (page 103).

[136] P. Sanders and C. Schulz. Distributed evolutionary graph partitioning. In *Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX)*, pp. 16–29, 2012. (page 2).

[137] J. Scharnow, K. Tinnefeld, and I. Wegener. Fitness landscapes based on sorting and shortest paths problems. In *Proceedings of the 7th International Conference Parallel Problem Solving From Nature (PPSN)*, Vol. 2439 of *Lecture Notes in Computer Science*, pp. 54–63. Springer-Verlag, 2002. (pages 62, 66).

[138] J. Scharnow, K. Tinnefeld, and I. Wegener. The analysis of evolutionary algorithms on sorting and shortest paths problems. *Journal of Mathematical Modelling and Algorithms*, 3:349–366, 2004. (pages 7, 15, 25, 35, 62, 63, 64, 65, 66).

[139] A. Schrijver. *Combinatorial Optimization – Polyhedra and Efficiency*. Springer, 2004. (page 36).

[140] M. Schulze. A new monotonic, clone-independent, reversal symmetric, and condorcet-consistent single-winner election method. *Social Choice and Welfare*, 36:267–303, 2011. (page 116).

[141] D. J. Sherratt. *Mobile Genetic Elements*. Oxford University Press, 1995. (page 31).

[142] P. Siarry and Z. Michalewicz, editors. *Advances in Metaheuristics for Hard Optimization*. Natural Computing Series. Springer-Verlag, 2008. (page 2).

[143] T. Storch. On the choice of the parent population size. *Evolutionary Computation*, 16:557–578, 2008. (page 7).

[144] T. Storch and I. Wegener. Real royal road functions for constant population size. *Theoretical Computer Science*, 320:123–134, 2004. (page 10).

[145] D. Sudholt. Crossover is provably essential for the Ising model on trees. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1161–1167. ACM Press, 2005. (page 10).

[146] D. Sudholt. Crossover speeds up building-block assembly. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 689–702. ACM Press, 2012. (pages 10, 11).

[147] A. M. Sutton and F. Neumann. A parameterized runtime analysis of evolutionary algorithms for the euclidean traveling salesperson problem. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1105–1111, 2012. (page 7).

[148] A. M. Sutton and F. Neumann. A parameterized runtime analysis of simple evolutionary algorithms for makespan scheduling. In *Proceedings of the 12th International Conference Parallel Problem Solving From Nature (PPSN)*, pp. 52–61, 2012. (page 7).

[149] E.-G. Talbi. *Metaheuristics: from design to implementation*. Wiley, 2009. (page 2).

• [150] M. Theile. Exact solutions to the traveling salesperson problem by a population-based evolutionary algorithm. In *Proceedings of the 9th European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP)*, Vol. 5482 of *Lecture Notes in Computer Science*, pp. 145–155. Springer-Verlag, 2009. (pages 16, 17, 25, 35, 36).

[151] G. Tsaggouris and C. Zaroliagis. Multiobjective optimization: Improved FPTAS for shortest paths and non-linear objectives with applications. *Theory of Computing Systems*, 45:162–186, 2009. (pages 69, 103).

[152] G. Tsaggouris and C. Zaroliagis. Multiobjective optimization: Improved FPTAS for shortest paths and non-linear objectives with applications. In

Proceedings of the 17th International Symposium Algorithms and Computation (ISAAC), Vol. 4288 of *Lecture Notes in Computer Science*, pp. 389–398, Berlin, Germany, 2006. Springer-Verlag. (pages 69, 103).

[153] L. G. Valiant. Evolvability. *Journal of the ACM*, 56, 2009. (page 3).

[154] V. Vassilevska, R. Williams, and R. Yuster. All-pairs bottleneck paths for general graphs in truly sub-cubic time. In *Proceedings of the 39th Symposium on Theory of Computing Conference (STOC)*, pp. 585–589. ACM Press, 2007. (page 116).

[155] S. Vassilvitskii and M. Yannakakis. Efficiently computing succinct trade-off curves. *Theoretical Computer Science*, 348:334–356, 2005. (page 103).

[156] K. Veeramachaneni, K. Vladislavleva, M. Burland, J. Parcon, and U.-M. O'Reilly. Evolutionary optimization of flavors. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1291–1298. ACM Press, 2010. (page 2).

[157] M. Wagner, K. Veeramachaneni, F. Neumann, and U.-M. O'Reilly. Optimizing the layout of 1000 wind turbines. In *Proceedings of the Annual Event of the European Wind Energy Association (EWEA)*, pp. 205–209, 2011. (page 2).

[158] M. Wagner, J. Day, D. Jordan, T. Kroeger, and F. Neumann. Evolving pacing strategies for team pursuit track cycling. In *Advances in Metaheuristics*, Vol. 53 of *Operations Research/Computer Science Interfaces Series*, pp. 61–76. Springer New York, 2013. (page 2).

[159] T. Walters. Repair and brood selection in the traveling salesman problem. In *Proceedings of the 5th International Conference Parallel Problem Solving From Nature (PPSN)*, Vol. 1498 of *Lecture Notes in Computer Science*, pp. 813–822. Springer-Verlag, 1998. (page 68).

[160] R. A. Watson and T. Jansen. A building-block royal road where crossover is provably essential. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1452–1459. ACM Press, 2007. (page 10).

[161] I. Wegener. Theoretical aspects of evolutionary algorithms. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP)*, Vol. 2076 of *Lecture Notes in Computer Science*, pp. 64–78. Springer-Verlag, 2001. (page 6).

[162] I. Wegener and C. Witt. On the optimization of monotone polynomials by simple randomized search heuristics. *Combinatorics, Probability and Computing*, 14:225–247, 2005. (page 14).

[163] C. Witt. Runtime analysis of the $(\mu + 1)$ EA on simple pseudo-Boolean functions. *Evolutionary Computation*, 14:65–86, 2006. (page 7).

[164] C. Witt. Worst-case and average-case approximations by simple randomized search heuristics. In *Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, Vol. 3404 of *Lecture Notes in Computer Science*, pp. 44–56, Berlin, Germany, 2005. Springer-Verlag. (page 7).

[165] G. J. Woeginger. When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)? *INFORMS Journal on Computing*, 12:57–74, 2000. (pages 15, 17, 18, 21).

[166] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on evolutionary computation*, 1:67–82, 1997. (page 6).

[167] A. H. Wright, M. D. Vose, and J. E. Rowe. Implicit parallelism. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1505–1517. Springer-Verlag, 2003. (page 9).

[168] T. Yu, L. Davis, C. M. Baydar, and R. Roy, editors. *Evolutionary Computation in Practice*, Vol. 88 of *Studies in Computational Intelligence*. Springer, 2008. (page 2).