Runtime Analysis of Evolutionary Algorithms with Complex Fitness Evaluation Mechanisms

Dogan Corus

November 27, 2017

Abstract

Evolutionary algorithms (EAs) are bio-inspired general purpose optimisation methods which are applicable to a wide range of problems. The performance of an EA can vary considerably according to the problem it tackles. Runtime analyses of EAs rigorously prove bounds on the expected computational resources required by the EA to solve a given problem. A crucial component of an EA is the way it evaluates the quality (i.e. *fitness*) of candidate solutions. Different fitness evaluation methods may drastically change the efficiency of a given EA. In this thesis, the effects of different fitness evaluation methods on the performance of evolutionary algorithms are investigated. A major contribution of this thesis is the first runtime analyses of EAs on bi-level optimisation problems. The performances of different EAs on The Generalised Minimum Spanning Tree Problem and The Generalised Travelling Salesperson Problem are analysed to illustrate how bi-level problem structures can be exploited to delegate part of the optimisation effort to problem-specific deterministic algorithms. Different bi-level representations are considered and it is proved that one of them leads to fixed-parameter evolutionary algorithms for both problems with respect to the number of clusters. Secondly, a new mathematical tool called the level-based theorem is presented. The theorem is a high level analytical tool which provides upper bounds on the runtime of a wide range of non-elitist population-based algorithms with independent sampling and using sophisticated high arity variation operators such as crossover. The independence of this new tool from the objective function allows runtime analyses of EAs which use complicated fitness evaluation methods. As an application of the level-based theorem, we conduct, for the first time, runtime analyses of non-elitist genetic algorithms on pseudo-Boolean test functions and also

on three classical combinatorial optimisation problems. The last major contribution of this thesis is the illustration of how the level-based theorem can be used to design genetic algorithms with guaranteed runtime bounds. The well-known graph problems Single Source Shortest Path and All-Pairs Shortest Path are used as test beds. The used fitness evaluation method is tailored to incorporate the optimisation approach of a well known problem-specific algorithm and it is rigorously proved that the presented EA optimises both problems efficiently. The thesis is concluded with a discussion of the wider implications of the presented work and future work directions are explored.

Acknowledgements

I am grateful to Per Kristian Lehre, for his guidance was most essential in navigating the field of runtime analysis to find my own research interests. It has been an invaluable experience to work with him. A very special gratitude goes to the Vice Chancellor's office in University of Nottingham which funded my PhD. I am grateful to my exam committee for their patience in giving me the unique opportunity to talk about my research for such an extended period of time. I owe my family gratitude for normalising the sustained state of madness that is post-graduate education. I would like to thank my friends for never asking how my thesis was going until they knew I would like to answer the question. Last but not the least, I would like to thank Birce; for she has never let me feel lost or alone while I was struggling to shape my future.

Contents

Chapter 1

Introduction

1.1 Motivation

Evolutionary computation (EC) draws inspiration from principles of Darwinian evolution for solving complex problems [32]. EC is often successfully applied for solving optimisation, machine learning and engineering design problems amongst many others [39]. A given environment is filled with a population of individuals that strive for survival and reproduction while competing for limited resources. The fitness of these individuals is determined by the environment and relates to how well they succeed in achieving their goals (i.e. their chances of surviving and reproducing). Darwin's insight was that small random variations, called *mutations*, occur during reproduction from generation to generation (i.e. variational inheritance) and the best individuals survive and reproduce (i.e. survival of the fittest) allowing evolution to progress [16].

Correspondingly, evolutionary algorithms (EAs) use these Darwinian principles, survival of the fittest and variational inheritance, to evolve high quality solutions of optimisation problems. In this setting, the individuals in the search space consist of candidate solutions for the problem that are used by selection and variation operators as "templates" to generate new solutions. Mutation and crossover (i.e. recombination) introduce variation in the population while selection operators use a fitness function to favour individuals with higher solution quality for survival. Ideally, the fitness function is designed such that the higher the fitness of an individual, the better it represents a solution to the problem.

Overall, evolutionary algorithms operate by iteratively sampling new solutions from a search space and updating a set of stored solutions

called a *population* [39]. In each *generation*, i.e., a single iteration of the EA, the *variation* operators (e.g. mutation and crossover) take the current population (often referred to as *parent population*) as input and sample a new set of solutions called the *offspring* population. The probability distribution over the search space from which the offspring population is sampled depends on the input population. Then the *selection* operator picks solutions according to fitness to form the population of the next generation.

The generality of the described process makes evolutionary algorithms *general purpose* methods which can be applied to various optimisation problems defined on various search spaces. All that is required is a way to represent candidate solutions as individuals of the population (i.e. the *representation*) and some measure of solution quality (i.e. the *fitness function*). Such a measure should allow comparison of any two solutions in the search space. As a result, EAs do not require any information about the problem structure apart from the fitness value of candidate solutions (i.e. they are *black-box optimisation algorithms* [30]).

Given such generality, one may wonder whether a given EA may be efficient for all problems defined on a given search space. In particular, could there be an EA which outperforms all others? The No Free Lunch Theorem [87] answers these questions by stating that all black-box randomised algorithms have the same expected performance averaged over all possible search problems. As a result, a given algorithm can only have an above average performance on restricted problem classes. The aim of performance analysis is to pair algorithms with the problem classes for which they perform well, and for which they perform poorly.

The most common approach for evaluating how well an EA performs on a class of problems is to execute it on a set of benchmark instances. However, through such experiments [39] nothing can be derived about how the performance of the algorithm is affected by slight changes in the parameters of the instance and/or of the algorithm. In particular, through empirical work it is not possible to derive performance results that hold for any instance of the problem class. Nor is it possible to compare different algorithms and parameter settings over the whole problem class.

An alternative methodology is to undertake a theoretical analysis of the evolutionary algorithm to assess its performance for the problem

class. Such an approach, commonly referred to as *runtime analysis*, aims to derive a relationship between the size of the problem and the time (i.e. runtime) required to find an optimal (or near optimal) solution [2].

The performance of EAs may vary considerably with different parameter settings. Obvious parameters that need to be set are population sizes, mutation and crossover rates (i.e. how often and how much variation) and selection probability distributions. Other crucial aspects affecting the performance of an EA are the choice of how to represent a solution and how to define the fitness function. In this thesis we will take the analytical runtime analysis approach to study the impact of different fitness functions on the performance of evolutionary algorithms in various contexts.

An optimisation problem generally consists of finding the input $x \in \mathcal{X}$ of an *objective function* $f : \mathcal{X} \to \mathbb{R}$ such that $f(x)$ is either minimal ($f(x) \leq f(y)$ for all $y \in \mathcal{X}$) or maximal ($f(x) \geq f(y)$ for all $y \in \mathcal{X}$) [74]. When applying an EA to optimise f, the most common way of defining the fitness function is to simply use the objective function f. This may be the only viable approach in black-box settings when no other knowledge is available . On the other hand, if more problem knowledge is available, then this may be exploited to produce more effective EAs. In particular, while the final goal is to find the solution optimising the objective function f, the EA may follow a fitness function which differs considerably from f. For instance, the fitness function may reward search points with low objective function values that however lead the search process "closer" to the optimum. Such a strategy may even facilitate avoidance of *local optima* [66] in the objective space which are "unseen" in the fitness function space. Herein, local optima are informally defined as solutions from which variation operators find it hard to produce solutions of higher fitness. Hence, using such alternative fitness functions may prevent deadlocks during evolutionary processes. Indeed, evolution *per se* does not aim to optimise. An organism only progresses when competition is available and once it has dominated its environment, progress is stalled until competition reemerges [38].

In some optimisation problems the objective function is only defined on a strict subset of the search space. Such problems are commonly referred to as *constrained optimisation* problems [32]. The search points where the function is defined are referred to as "feasible" solutions. The

remaining search points are called "infeasible". When the infeasible region is large, most newly generated solutions are discarded and the algorithm is forced to conduct a blind search between "islands" of feasible solutions in the objective function space. Using extra available problem knowledge, a fitness function may guide the search through infeasible subspaces [70, 37, 65, 3].

Overall, in this thesis we rigorously analyse the effects in terms of performance of increasing the amount of problem knowledge or changing the kind of problem knowledge used by the fitness function to guide the EAs in their evolutionary search.

We tackle this question in various contexts. First, we consider simple, so-called elitist EAs using only one individual (i.e. *(1+1) EA*) in the context of hard combinatorial optimisation problems. Then, we move our attention to more sophisticated population-based EAs using crossover and stochastic selection operators (i.e. standard *genetic algorithms* [39]). To achieve this ambitious goal, we design an analytical framework that provides upper bounds on the runtime of genetic algorithms. Afterwards, we design a genetic algorithm for solving shortest path problems and use the analytical framework to derive rigorous bounds on its runtime.

1.2 Overview of the Thesis

The first part of the thesis considers a simple (1+1) EA for two NP-hard problems, Generalised Minimum Spanning Tree [68] and Generalised Travelling Salesperson Problem [41]. Since it is not expected that any algorithm efficiently optimises NP-hard problems we will evaluate algorithmic performance using standard notions from parameterised complexity [35]. To appreciate changes in performance we compare two different fitness evaluation methods. These methods rely on a partition of the search space into subsets called *upper level search points*. In this context, the fitness of an upper level solution is the objective value of the best solution in the corresponding subset. The goal of the EA is to traverse the upper level search space by querying the least number of search points. Such an approach is commonly known as *bi-level optimisation* [6]. Different partitionings of the search space induce different fitness functions.

The rest of the thesis considers standard genetic algorithms using

stochastic selection and crossover. First, we introduce a technique to achieve upper bounds on the runtime of genetic algorithms, by extending and generalising a common technique used in runtime analysis to obtain upper bounds on the runtime of EAs called *artificial fitness-levels* method [84]. A requirement of this traditional method is that the EA uses elitism such that the best solution is never removed from the population. Recently, the methodology was extended to deal with several non-elitist population-based EAs [60, 12]. One considerable limitation of the latter technique is that it only applies to algorithms using unary variation operators such as mutation, hence does not allow the achievement of upper bounds on the runtime of EAs using more sophisticated operators such as crossover. In this thesis we present a new level-based method that allows the analysis of a wide class of EAs including standard genetic algorithms. A novelty of the presented methodology is that it allows to define levels of the search space which are independent of the objective function.

In the final part of this thesis, we apply our newly introduced methodology to analyse a genetic algorithm for the well-known shortest path problem. To appreciate the power of the new analytical framework, we equip the GA with a fitness function that only uses comparisons between search points rather than returning explicit function values and provide a rigorous runtime analysis.

The major contributions of this thesis are the following.

- The first runtime analysis of an evolutionary algorithm on a bi-level optimisation problem is provided. Runtimes of different evolutionary algorithms on the generalised minimum spanning tree problem and the generalised travelling salesperson problem are analysed. Fixed-parameter tractable evolutionary algorithms are presented for both of the problems. Complementary abilities of different solution representations and fitness evaluation methods are shown. Sufficient conditions for the solution representation of a bi-level problem that allow a fixed-parameter tractability result are provided.

- A general theorem which proves upper bounds on the runtime of non-elitist evolutionary algorithms with independent sampling is provided. This level-based theorem covers a larger class of al-

gorithms than its predecessors. Upper bounds on the runtime of non-elitist genetic algorithms on various problems are proved using this new tool including the standard sorting problem.

- A genetic algorithm is designed with the guidance of the level-based theorem for the shortest path problem. The requirements of this analytical tool are translated into design decisions leading to an algorithm with a proven upper bound on its runtime. The performance of the GA is analysed on the shortest path problem and it is rigorously proved that it is efficient.

- This is the first time a GA, using stochastic selection and crossover, is analysed on a standard combinatorial optimisation problem.

This thesis is structured as follows. Chapter 2 provides basic definitions, algorithmic schemes, commonly used algorithmic components and analytical methods that will be used in the rest of the thesis. Chapter 3 discusses the relevant literature placing the work presented in the rest of the thesis in context. In Section 3.1, prior parameterised complexity results in the field of evolutionary algorithms are discussed. Section 3.2 overviews the state of the art concerning the techniques used in the runtime analysis of population-based EAs. Section 3.3 surveys prior runtime results of evolutionary algorithms for shortest path problems.

The original contributions of the thesis will be presented in Chapters 4 to 6. Chapter 4 presents the runtime analysis of the two bi-level optimisation problems. Parameterised runtime analysis results are presented for both GMSTP and GTSP. Two different representations for the GMST problem are investigated using different upper and lower level problems. We show that the representation can make the difference between an EA that is a fixed-parameter tractable EA and one that is not. This algorithm will then be adapted for GTSP and proved to be a fixed-parameter tractable EA for the problem.

In Chapter 5 a new analytical method for the runtime analysis of non-elitist population-based algorithms called *level-based analysis* is provided. In Section 5.4, a corollary to the main theorem for the analysis of genetic algorithms will be applied to some pseudo-Boolean benchmark problems and to the sorting problem. Since level-based analysis allows tracking the progress of the search process over arbitrarily defined lev-

els without referring to a fitness function, we will use this theorem in our final contribution chapter.

As a final contribution we design a genetic algorithm for solving the single source and all-pairs shortest path problems in Chapter 6. The design process will be guided by the requirements of the level-based theorem presented in Chapter 5. We will use a solution comparison method that does not have an explicit fitness function, thus mimicking the well-known Dijkstra algorithm for the shortest path problem.

In Chapter 7, we will review the major contributions of the thesis and point out interesting future directions.

1.3 Publications

Chapters 4-6 are each derived from a different publication. The work presented in Chapter 4 on the GMST problem was derived from the results that have been published in the 2012 ACM Genetic and Evolutionary Computation Conference (GECCO'12) [9] and won the best paper award at the conference in the "Combinatorial optimisation" track. This work was later extended with GTSP analysis to a journal paper published in Evolutionary Computation [10]. The majority of my contributions to these publications was focused on the analysis of the GMST problem. The joint work with Duc-Cuong Dang, Anton Eremeev and Per Kristian Lehre from which the Chapter 5 was derived was published in the proceedings of the 13th International Conference on Parallel Problem Solving from Nature (PPSN 2014) [8]. Besides collaborating with my co-authors to establish the main result of the paper, I particularly contributed in devising the corollary of the level-based theorem which is applicable to genetic algorithms, and applying it to sorting problems. Finally, Chapter 6 uses the material presented in the 11th Metaheuristics International Conference (MIC'15) as a joint work with Per Kristian Lehre. An extended version of the conference paper is under review to be published as a chapter of "Recent Developments in Metaheuristics" by Springer.

Chapter 2

Preliminaries

In this chapter we will introduce common algorithmic components of evolutionary algorithms, present the mathematical framework for runtime analysis and introduce common problems and methods used in the theory of evolutionary algorithms (EAs). In Section 2.1, a general scheme for EAs is introduced with its main components. In Section 2.2 the runtime of an EA is formally defined. In Section 2.3 the optimisation problems considered in this thesis are introduced. An overview of the most common analytical methods used in the analysis of EAs is given in Section 2.4.

Throughout the thesis the following notation is used. For any positive integer n, we define $[n] := \{1, 2, ..., n\}$. For a population P of size μ, both P^i and $P(i)$ for $i \in [\mu]$ denote the ith solution in the population. The natural logarithm is denoted by $\ln(\cdot)$. The complement of an event $\mathcal{E}$ is denoted by $\bar{\mathcal{E}}$.

2.1 Evolutionary algorithms

Evolutionary algorithms use Darwinian principles to solve optimisation problems. They are randomised algorithms which incorporates the randomness into the way the algorithm chooses new search points to evaluate. However, they differ from most randomised algorithms by being a general purpose optimisation algorithm. Overall, EAs iteratively sample new solutions and update their list of current solutions called *population* (Algorithm 1). In each iteration of the EA, first the first selection operation (i.e., *the parent selection*) is applied to the population to determine the parent population (P') which will be perturbed by the variation op-

erator. The *variation* operator takes the parent population (P') as input and samples a new set of solutions P'' (the offspring population) from a probability distribution that depends on P'. The final *selection* operator (i.e., environmental selection) then samples the population of the next generation amongst the individuals of the parent and the offspring populations [39].

Algorithm 1 Evolutionary Algorithm

1: Initialise population P
2: **while** termination condition not satisfied **do**
3: $P' \leftarrow selection(P)$
4: $P'' \leftarrow variation(P')$
5: $P \leftarrow selection(P, P'')$
6: **end while**

The above algorithm is given without a specific termination condition. While in practice the termination condition is a fixed iteration limit, most theoretical analyses considers the runtime as the number of fitness evaluations until the optimum solution is found for the first time. Although the runtime until the optimum is found allows the performance analysis of the algorithm independent of any termination conditions, its application is only possible for problems whose optimal value is known beforehand.

The following two sections provide an overview of common variation and selection operators.

2.1.1 Variation Operators

Variation operators take an input population and produce an offspring population. In order to conserve and exploit the information carried over by the population, the offspring population is sampled to be somewhat similar to the input population with small random changes prescribed by a variation operator. One of the distinctive parameters of a variation operator is its arity. While *mutation* operators accept a single solution and perturb it, *recombination* operators use information from multiple solutions to produce an offspring solution. Different solution representations require different variation operators. For the sake of this thesis new operators will be defined when necessary and common elementary variation operators will be presented in this section [73].

Definition 1. *A mutation operator is said to be global if the random offspring solution $mut(x)$, given the input solution x, satisfies*

$$\forall (x, y) \in \mathcal{X}^2 \quad \Pr\left(mut(x) = y\right) > 0 \tag{2.1}$$

Global operators allow the algorithm to find the optimal solution with probability one in finite time [76]. In contrast, *local* operators have a *neighborhood*, a set of solutions that the mutation operator can output. An EA with a local operator might get stuck on *local optima*, solutions whose neighborhood does not contain a better solution.

For bitstring representations of a solution, $x \in \{0, 1\}^n$, the most commonly used global operator is *Standard Bit Mutation* (Algorithm 2) .

Algorithm 2 Standard Bit Mutation (SBM)

for $i := 1$ to n **do**
  With probability χ/n set $x\,[i] := 1 - x\,[i]$.
end for

The parameter χ characterises an SBM operator and is equal to the expected Hamming distance between the parent and the offspring solution. A similar mutation operator which is not global is the k-bit flip operator. It dictates exactly how many bits will be flipped in the offspring solution and picks the location of the bits uniformly at random without replacement.

Recombination operators are variation operators that use information from multiple parents to create an offspring. The evolutionary algorithms that use recombination operators are called *genetic algorithms*. Recombination operators used in evolutionary algorithms are inspired by the crossover that happens during the *meiosis* of living cells [85]. By crossover genetic information from both parents are transferred to the offspring so that the offspring solution can embody the strengths of both parents.

Most common recombination operators are named after this natural phenomenon. An elementary crossover operator for bit-string representation is *uniform crossover* which picks each bit value from one of the parents with probability $1/2$ (Algorithm 3).

Another crossover operator for bit-string representation is k-point crossover. It uniformly selects k bit positions and the bit-string of parents are broken down at these positions into $k+1$ segments. Two offspring so-

Algorithm 3 Uniform Crossover

for $i := 1$ `to` n **do**
  with probability $1/2$,
    Set $z[i] := x[i]$
  otherwise
    Set $z[i] := y[i]$
end for

lutions are produced by combining alternating segments of both parents. A widely used special case of k-point crossover is one-point crossover which divides the parent bit-strings into two segments at a random point uniformly distributed along the bit-string and recombines these halves.

2.1.2 Selection Operators

In its most general definition, given a set of solutions, a selection operator returns a subset of the given solutions [73]. Selection operators vary according to the sizes of the input and output solution sets and how the operator incorporates the information from the fitness function. A high-level categorization divides selection operators into *elitist* and *non-elitist* selection operators. Elitist selection guarantees that at any point during the run of the algorithm the fittest solution sampled so far is kept in the population. Non-elitist selection operators have a positive probability of discarding the best solution in order to avoid getting stuck on local optima. Nevertheless, non-elitist selection still probabilistically favours solutions with higher fitness and can provide sustained increase in the solution quality. The degree of elitism is measured according to the *selective pressure* of the operator [63] or its *takeover time* [40]. The selective pressure $\beta(\gamma)$ of a selection operator is the probability that selected solutions are among the best γ solutions of the input population. The takeover time is the expected number of selection operations before the population only consists of solutions with the highest fitness value. According to their purposes described in Algorithm 1, the selection operators of an evolutionary algorithms fall into one of the two categories: parent selection and environmental selection. The first selection operation, parent selection, selects the subset of population which will be the input of the variation operation. It determines the exploration direction of the algorithm by concentrating the next batch of new search points to be evaluated around the chosen parent population. Parent selection

operators mostly allow an individual in the population to be selected more than once as a parent, allowing it to create multiple offsprings in the next generation. This effect allows the number of fitter individuals to increase over time. The same progress is more directly achieved by the second selection operation, environmental selection, which simply removes the low-fitness individuals from the population or equivalently selects the high-fitness individuals which will be carried over to the next generation.

$(\mu + \lambda)$ (Truncation) Selection

$(\mu + \lambda)$ is one of the most commonly used *elitist* selection operators in evolutionary algorithms [17]. The selection mechanism takes a set of solutions of size $\mu+\lambda$ (that consists of μ parental and λ offspring solutions) as input and returns the μ solutions with the highest fitness values. This implies that λ offspring solutions are sampled at every generation and pooled with the μ solutions carried over from the previous generation. Ties may be dealt with in various ways. One common strategy is to break ties uniformly at random with or without favouring offspring over parents

A special case of $(\mu+\lambda)$ selection is the (1+1) selection operator which carries over a single individual from generation to generation. The (1+1) EA is the most widely analysed algorithm in the theory of evolutionary algorithms due to its simplicity. In the (1+1) EA ties are broken by preferring the offspring over the parent.

(μ, λ)-Selection

Given an input of λ offspring solutions, (μ, λ)-selection returns the best $\mu \leq \lambda$ solutions [60]. Unlike $(\mu + \lambda)$-selection, the input population does not necessarily contain the parent solutions. It is a non-elitist selection operator because there is a non-zero probability that the λ offspring solutions will all have a worse fitness than the best parent solution.

Ranking selection

Ranking selection sorts the input population according to fitness function values and assigns to each individual a probability proportional to

its rank [60]. The operator then selects a solution according to the probabilities that are assigned. The individuals are ranked from 0 to 1, 0 being the rank of the best individual. The assigned probabilities are characterised by a positive function $\alpha : \mathbb{R} \to \mathbb{R}$ that satisfies $\int_0^1 \alpha(y)dy = 1$. The cumulative probability of selecting a solution of rank γ or higher is set to $\int_0^y \alpha(\gamma)dy$. A common instance of ranking selection is *linear* ranking where the ranking function is $\alpha(\gamma) = \eta(1 - 2\gamma) + 2\gamma$ where η adjust the selective pressure.

k-Tournament Selection

The k-tournament selection operator picks k solutions uniformly at random with replacement from the input population and selects the solution with the highest fitness value among them. The selective pressure of the operator increases with higher k since the best of a larger group is selected. Since the solutions are picked with replacement it is a non-elitist operator regardless of the value of k.

2.2 Runtime

The computational resources that randomised algorithms require vary according to the random choices made during their runs. The performance measures are therefore expressed as expected values or with probability bounds. The theoretical analysis of randomised algorithms are conducted with a wide array of tools from probability theory. One of the well studied objects of probability theory, Markov chains [53], can model most evolutionary algorithms.

Definition 2. *A Markov chain is a sequence of random variables* $(X_1, X_2, \ldots, X_t, \ldots)$ *that for all* $t > 0$ *satisfies:*

If $\Pr(X_1 = x_1, X_2 = x_2, \ldots, X_{t-1} = x_{t-1}) > 0$*, then*
$\Pr(X_t = x | X_1 = x_1, X_2 = x_2, \ldots, X_{t-1} = x_{t-1}) = \Pr(X_t = x | X_{t-1} = x_{t-1})$
for all $t \in \mathbb{N}$ *and* $x_1, x_2, \ldots, x_t \in S$ *.*

A Markov chain is characterised by a state space S and a transition matrix M. The state space is the set of values that the random variable X_t can have. One can define the transition matrix as the probabilities $m_{ij} = \Pr(X_{t+1} = s_j | X_t = s_i)$ for each state pair $(s_i, s_j) \in S^2$.

We use Markov chains to model the sequence of generations/populations of the EAs. Let the Markov chain $(P_1, P_2, \ldots, P_t, \ldots)$ represent the population of an evolutionary algorithm. For an EA with fixed population size λ, operating on a finite search space $\mathcal{X}$; the state space S of P_i is $\mathcal{X}^\lambda$, the set of all possible populations. The variation and selection operators determine the transition matrix.

In the case of evolutionary algorithms the time unit is the generation. However, the different complexity of operators prevents comparing the computational complexity of two evolutionary algorithms in terms of number of generations. If the selection operator relies on a *fitness function*, $f : \mathcal{X} \to \mathbb{R}$, when comparing solutions, then the runtime can be expressed in terms of number of fitness function evaluations. The runtime in terms of fitness function evaluations for an EA with fixed offspring population size λ is calculated by multiplying the runtime in number of generations with λ, since λ new solutions are evaluated in every generation [2].

Let $\mathcal{X}_{opt} \subseteq \mathcal{X}$ be the so-called target subset of the search space and $\mathcal{X}_{sub} := \mathcal{X} \setminus \mathcal{X}_{opt}$ (i.e. set of optimal and suboptimal solutions respectively). Then we can divide the state space of P_i into two similar subsets: The suboptimal states $S_{sub} := \mathcal{X}_{sub}^\lambda$ (i.e. the set of populations which consists of λ suboptimal solutions), and the optimal states $S_{opt} := S \setminus S_{sub}$. Building on this representation of evolutionary algorithms we can easily provide a formal definition of the runtime for an EA with constant population size λ [45].

Definition 3. *The runtime, $\mathcal{T}$, of an evolutionary algorithm with population $(P_t)_{t\in\mathbb{N}}$ of size λ is:*

$$\mathcal{T} := \lambda \min \{t | P_t \in S_{opt}\}$$

The runtime of a randomised algorithm is a random variable itself. Most often, we are interested in its expectation. Alternatively, a stronger result can provide a bound on the runtime which will hold with a specific probability, *i.e.* $\forall k \Pr(\mathcal{T} \leq k) \leq f(k)$ for some function f.

2.2.1 Parameterised Runtime

Parameterised runtime analysis of an algorithm provides the relationship between the performance of the algorithm with respect to a parameterisation κ of the problem [28]. The parameterisation κ is a function

which maps any instance I of the problem to a real value $k := \kappa(I)$. This parameterised runtime can show that an algorithm optimises a problem efficiently (*i.e.* in polynomial time with respect to the input size) if k has certain values and inefficiently (*i.e.* in exponential time with respect to the input size) otherwise. If an algorithm $\mathcal{A}$ optimises a problem with input size n in $poly(n)$ time for any fixed value of k, then the parameterised problem is said to be in the class FPT and the algorithm $\mathcal{A}$ is called an FPT-algorithm. The runtime of a fixed-parameter tractable algorithm has the form $f(k) \cdot poly(n)$ where the first multiplicative factor is a recursive function, $f(k)$ of k, which is independent of the input size n and the second factor is a polynomial of n. A similar but weaker result is obtained if the runtime is of order $\mathcal{O}(n^k)$. Such an algorithm is called an XP-algorithm and the problem is said to be in the problem class XP. Parameterised analysis in the context of evolutionary algorithms will be presented in Section 3.1.

2.3 Problems

The runtime of the algorithms are also used for classifying problem hardness through complexity classes. The complexity class P consists of decision problems which can be solved deterministically in polynomial time. The complexity class NP is the class which consists all the problems whose solutions can be *tested* deterministically in polynomial time. Even though it is clear that $P \subseteq NP$, it is a central unanswered question of theoretical computer science whether $P = NP$ or $P \neq NP$. A subclass of NP problems called NP-complete consist of the hardest problems in NP which are all polynomially reducible to each other. Cook's Theorem [7] implies that if any problem in the NP-complete class is solved by a deterministic algorithm in polynomial time, then all problems in NP are polynomially solvable. Optimisation problems are subject to a similar classification such that if the decision version of an optimisation problem is in the class NP-complete then the optimisation version of the problems is said to be NP-hard.

Heuristic methods like evolutionary algorithms are generally used to solve NP-hard problems [67] since the problems in P can be solved exactly by deterministic polynomial-time algorithms. However, early work on runtime analysis of evolutionary algorithms has focused on

simple pseudo-Boolean functions since harder problems required more complex analytical tools. The most widely used functions are ONEMAX and LEADINGONES.

Definition 4. *The functions* ONEMAX *and* LEADINGONES *are defined for any bit-string* $x \in \{0,1\}^n$ *as:*

$$\text{ONEMAX}(x) := \sum_{i=1}^{n} x_i \tag{2.2}$$

$$\text{LEADINGONES}(x) := \sum_{i=1}^{n} \prod_{j=1}^{i} x_j \tag{2.3}$$

Later works expanded the scope of runtime analysis of evolutionary algorithms to classical combinatorial optimisation problems [71]. In the following section, we will describe some of the important problems that will be discussed in this thesis.

2.3.1 The generalised minimum spanning tree problem

The NP-hard generalised minimum spanning tree problem (GMSTP) is given by an undirected complete graph $G = (V, E, c)$ on n nodes with a cost function $c\colon E \to \mathbb{R}^+$ that assigns positive costs to the edges. Furthermore, a partitioning of the node set V into m pairwise disjoint clusters $V_1, V_2, \ldots, V_m$ is given such that $n = \sum_{i=1}^{m} |V_i|$ [68]. A GMSTP with $|V| = m$ corresponds to an instance of the regular minimum spanning tree problem.

A solution to the GMSTP problem consists of two components, the m chosen nodes P, called *the spanned nodes*, in the m clusters, and a minimum spanning tree T on the graph induced by the spanned nodes. More precisely, a solution $S = (P, T)$ consists of a node set $P = (p_1, \ldots, p_m) \in V^m$, where $V^m = V_1 \times V_2 \times \cdots \times V_m$ and a spanning tree $T \subseteq E$ on the subgraph $G[P] = G(P, \{e \in E \mid e \subseteq P\})$ induced by P. The cost of T is the cost of the edges in T, i. e.,

$$C(T) = \sum_{(u,v)\in T} c(u, v).$$

The goal is to compute a solution $S^* = (P^*, T^*)$ which has minimal

cost among all possible solutions $S = (P, T)$. For an easier presentation, we assume in some cases that edge costs can be ∞. In this case, we restrict our investigations to solutions that do not include edges with cost ∞. Alternatively, one might view this as the GMSTP defined on a graph that is not necessarily complete.

The GMSTP is fixed-parameter tractable since there is an exact algorithm which solves it in $\mathcal{O}(m^{m-1}n^2)$ time [75]. This runtime can be divided into two multiplicative factors: the number of possible trees over the cluster graph (m^{m-1}) and the time it takes to optimise the selection of nodes given a specific tree on the cluster graph (n^2).

2.3.2 The generalised traveling salesperson problem

An instance of the NP-hard generalised travelling salesperson problem (GTSP) is given as a complete graph $G = (V, E, c)$ with a cost function $c\colon E \to \mathbb{R}^+$ and a partitioning of the node set V into m clusters V_i, $1 \leq i \leq m$. A solution to the problem is formulated in a similar manner to GMSTP. A single node is picked from each cluster V_i and this set of spanned nodes P induces a subgraph $G[P]$. The goal is to find a Hamiltonian cycle π of minimal total cost $C(\pi) = \sum_{(u,v)\in\pi} c(u, v)$. As we noted about GMSTP, an incomplete graph can be represented by assigning infinite costs to missing edges and an instance where $|V| = m$ corresponds to a regular TSP problem.

2.3.3 The shortest path problem

An instance of a shortest path problem is given as an input graph, $G = (V, E)$, with a cost function $c\colon E \to \mathbb{R}^+$ and a set $S \subseteq V^2$ of pairs of vertices. The objective is to find a *path* (sequence of edges) $P^{u,v}$ that connects each vertex pair $(u, v) \in S$ while minimising the total weight $\sum_{(u,v)\in S} \sum_{(w,z)\in P^{u,v}} c(w, z)$. The Single Source Shortest Path (SSSP) problem and All-Pairs Shortest Paths (APSP) Problem are the most studied versions of shortest path problems in the context of EAs. In SSSP, the set of vertex pairs to be connected, is $S = \bigcup_{v\in V\setminus\{s\}}(s, v)$ for a source vertex $s \in V$ which is given as part of the input. For the all-pairs version $S = \{(u, v) \in V^2 | u \neq v\}$ which means that all the vertex pairs should be connected. Both versions of the problem are polynomially solvable (in the complexity class P).

2.4 Methods

Various methods for analysing EAs were developed to tackle increasingly more complex settings. While some methods like artificial fitness levels, are restricted to the context of evolutionary algorithms, many others like drift analysis and concentration bounds were borrowed from probability theory.

2.4.1 Artificial fitness levels

A function partitions its domain into *fitness levels*, subsets of the search space that consist of search points with the same function value. This partition is useful in the analysis of elitist evolutionary algorithms since once the algorithm obtains a solution from a certain fitness level, the subsequent generation will retain that solution until a better fitness level is discovered. This allows the partitioning of the search process into epochs, each starting with the discovery of a solution from a fitness level and lasting until a solution from a higher (for maximisation, lower for minimisation) fitness level is discovered [2].

For a search space $\mathcal{X}$ with the fitness function $f : \mathcal{X} \to \{R_1, R_2, \ldots, R_m\} \subset \mathbb{R}$, where the values R_i satisfy $R_i < R_{i+1}$ for all $i \in \{1, 2, \ldots, m\}$,the *canonical* fitness level partition of the search space is the collection of subsets $\{L_1, L_2, \ldots, L_m\}$ of $\mathcal{X}$ such that $L_i = \{x \in \mathcal{X} | f(x) = R_i\}$. The probability s_i, that any solution from the L_i will be mutated into a solution from $\bigcup_{j=i+1}^{m} L_j$ is called the *improvement probability*. For an elitist (1+1) EA [29], the expected runtime of each epoch can be bounded from above by $1/s_i$ and the total expected runtime can be bounded from above by the sum $\sum_{i=1}^{m} 1/s_i$.

Artificial fitness levels ($<_f$ partitions) are obtained by combining multiple levels into a super-level [2]. For two sets $A, B \subseteq \mathcal{X}$ and a function f which is defined on both A and B, the relation $A <_f B$ holds if for all $a \in A$ and $b \in B$, $f(a) < f(b)$ holds. A $<_f$ partition of the search space is a collection of disjoint non-empty subsets $(A_1, \ldots, A_m)$ of $\mathcal{X}$ such that $A_i <_f A_{i+1}$. The artificial fitness levels are commonly used to combine fitness levels where the improvement probability is higher in order to minimise $m \cdot \max_{i \in [m]} (1/s_i)$ which is an upper bound on the runtime.

2.4.2 Tools from probability theory

As mentioned above, evolutionary algorithms are stochastic processes which are the subject of probability theory. Many tools developed for the analysis of stochastic processes are applicable to the runtime analysis of evolutionary algorithms.

An important tool imported from probability theory is drift analysis. Drift analysis relates the expected improvement towards finding the optimal solution at each generation to the expected time until the optimal solution is found. This relationship is built on a so-called *distance function*.

Definition 5. *A distance function is any function* $d : \mathcal{X}^\lambda \to \mathbb{R}_0^+$ *which satisfies*

$$d(P) = \begin{cases} 0 & \text{if } P \in S_{opt}, \\ > 0 & \text{otherwise.} \end{cases}$$

This function characterises the state of a population P with a single variable, which represents the population's distance from the optimal states. Then, the single-step behaviour of a Markov chain can be quantified as the expected change in the distance. The only requirement for the distance function is that it maps the optimal states to zero and all other states to a positive value. However, for the distance function to be useful in the analysis, the distance of a state should reflect the expected runtime of the algorithm starting from that initial state. By bounding the *drift*, the expected single-step distance change, of a Markov chain, the expected long-term progress can be bounded as well .

Theorem 6 (Drift Theorem [42])**.** *Let* $X_1, X_2, \ldots$ *be a stochastic process over* S, $d : S \to \mathbb{R}_0^+$ *be a distance function on* S. *Define* T *to be the first time* t *such that* $d(X_t) = 0$. *If there exist* $B, \Delta > 0$ *such that,*

1. $\forall t \geq 0 : \Pr\left(d(X_t) < B\right) = 1$, *and*
2. $\forall t \geq 0 : E\left[d(X_t) - d(X_{t+1}) \mid X_t\right] \geq \Delta$,

then $E\left[T\right] \leq B/\Delta$.

An important subset of the analytical tools provided by probability theory relates to the relationship between the expected value of a random variable and the actual probability of deviating from that expectation. One such elementary result is Markov's Inequality.

Theorem 7 (Markov's Inequality)**.** *Let X be a random variable assuming only non-negative values. Then for all $t \in \mathbb{R}^+$,*

$$\Pr(X \geq t) \leq \frac{E[X]}{t}.$$

A similar inequality which provide exponential probability bounds is Chernoff's Inequality for binomial identically distributed independent random variables.

Theorem 8 (Chernoff's Inequality [31])**.** *Let $X_1, X_2, \ldots, X_n$ be independent random variables each defined on $\{0, 1\}$, then $X = \sum_{i=1}^{n} X_i$ satisfies:*

$$\Pr(X \leq (1-\varepsilon)E[X]) \leq \exp\left(-\varepsilon^2 E[X]/2\right) \tag{2.4}$$

$$\Pr(X \geq (1+\varepsilon)E[X]) \leq \exp\left(-\frac{\varepsilon^2}{2+\varepsilon}E[X]\right) \tag{2.5}$$

for any $\varepsilon \in [0, 1]$.

.

Chapter 3

Literature Review

In this chapter the relevant literature is discussed to provide a background for our research. The following sections provide an overview of parameterised complexity results in the field of evolutionary algorithms, the state of the art techniques used in the runtime analysis of population-based EAs, and runtime results of EAs on shortest path problems.

3.1 Parameterised complexity in Evolutionary algorithms

EAs are used in practice to find solutions of acceptable quality to instances of hard optimisation problems. The theory of EAs naturally aims at covering the application area of EAs. The initial runtime results of randomized algorithms on hard combinatorial optimisation problems aim at various goals like reducing the exponent of the input size in the runtime, polynomial time approximations, and average case analysis [71]. Another possible venue for tackling the runtimes of hard combinatorial problems is parameterised analysis which was adopted relatively late by the theory of EAs in 2010s. In this section we will introduce basic notions of parameterised complexity analysis and discuss how it was integrated with the theory of EAs.

The runtime of an algorithm is generally expressed as a function of the input size. The dependence on input size is necessary, because the main concern of the algorithm designer is whether the algorithm is scalable to large problem instances that occur in real life. However, the input size is not the only quantifiable characteristic of a problem instance.

Other measures, functions that map any domain to real numbers, can be defined for problems in order to characterize a problem instance. Each measure defined on the instance space is called a parameter and a runtime that involves parameters other than the input size is called *parameterised complexity*. For any parameter k, if there exists an algorithm which solves any instance of a problem of size n in $poly(n) \cdot f(k)$ time for an arbitrary function $f(k)$, the problem is said to be in class FPT (*fixed-parameter tractable*) and the algorithm is called a *fixed-parameter tractable algorithm*.

Fixed-parameter tractability (FPT) is a significant result if the parameter k is particularly small in real-life instances. Because the FPT result then implies that the problem is practically polynomially solvable. An early example of a fixed-parameter tractable problem is the type checking problem which can be solved in $\mathcal{O}(2^k n)$ where k is the maximum nesting depth of the type declarations. In most cases k is less than 10 [64]. Certain games which are paremeterised by the number of winning moves in a winning strategy can be solved by an XP-algorithm [1]. Fixed-parameter tractability has been a vibrant research area of classical computer science, which came before the theory of EAs. It is an interesting direction when trying to show that EAs tend to 'work in practice'. Concepts from FPT can be transferred easily to the theory of EAs by considering the expected runtime. An EA is an FPT-EA if it can solve a parameterised problem (L, κ) in expected $f(k) \cdot poly(n)$ function evaluations [55].

The theory of evolutionary algorithms lack the tools to explain its efficiency on some hard combinatorial optimisation problems. Their practical use cannot be rigorously proven with polynomial upper bounds on the expected runtime for NP-hard problems. The parameterised analysis reflects the relationship between the runtime of algorithms and the quantifiable characteristics of the problem instances. This new dimension of analysis allow FPT algorithms which solve an NP-hard problem in polynomial time if a particular parameter of the problem instance is fixed. Using parameterised analysis, FPT-EAs are presented for problems in the class FPT such as Planar Euclidean Travelling Salesperson Problem, VERTEX COVER, MAXIMUM LEAF SPANNING TREE, MAKESPAN SCHEDULING, MAX-2-SAT and its special case MAX-(2,3)-SAT [80, 69, 81, 55, 56, 82, 79, 36].

3.1.1 Parameters

The most important insight that a fixed-parameter tractability result can provide about a problem is which parameters the exponential complexity can be isolated to. The value of this parameter then determines the computational effort required to solve the problem, and guides the search for an optimal solution or an approximation. The most common parameter for FPT results in evolutionary algorithms so far is the "standard parameterisation" which is used for minimum VERTEX COVER, MAXIMUM LEAF SPANNING TREE, MAKESPAN SCHEDULING and MAX-2-SAT problems [55, 56, 82, 79, 36].

The standard parameterisation of a combinatorial optimization problem is (assuming minimization), given an instance and a parameter k, "Is the value of the optimal solution at most k"? While the parameter k is explicit in the decision version of optimisation problems, for the optimisation version it is the value of the optimal solution itself. An FPT-EA which uses the standard parameterisation is efficient if the sought after optimal solution is small.

An important caveat for some of the standard parameterisation FPT-EAs is that when the recursive function $f(k)$ from the definition of FPT is $\Omega(2^n)$, even though the upper bound on the runtime can be expressed as a function of another parameter, the bound actually depends exponentially on n [82, 79, 56]. When an FPT result depends exponentially on n, the corresponding parameter does not provide a significant insight about the hardness of the instance. In most cases such results are complemented by FPT results with alternative parameterisations [82, 79].

Apart from the standard parameterisation, few other parameters appear in FPT-EAs . Most of the other parameters are unique to the respective problems. Two more parameters are suggested for the MAKESPAN SCHEDULING problem [82]. The first one is the optimal discrepancy, which is defined as the difference between the loads of the machines (discrepancy) in the optimal solution. The second parameter is the optimal critical path size, the number of jobs scheduled on the fuller machine.

The fixed-parameter in the Planar Euclidean Travelling Salesperson Problem (PETSP) is the number of cities that are not on the convex hull of the cities. This dependency of hardness to the number of "inner" points is due to the requirement that any optimal tour should have the vertices

on the convex hull in the same order as they appear on the convex hull [80, 69, 81].

The MAX-(2,3)-SAT problem is a restricted class of MAX- 2-SAT where each variable appears in at most three clauses of the formula. The instance graph is obtained by adding a node for each literal and adding an edge between each literal that appears in the same clause. The parameterisation used in the analysis is the diameter (maximum shortest path distance between any two nodes) of the variable graph [79].

The most important aspect of a fixed parameter tractability result is the parameter used. For FPT-EAs, the ubiquitous parameter is the standard parameterisation which is the parameter introduced in the decision versions of optimisation problems. Apart from standard parameterisation, problem specific parameters are used in some results.

3.1.2 Kernelization

Another characteristic of an FPT-EA besides fixed-parameter is its progress on the solution space over time. For some FPT-EAs, the first stages of this progress are similar to the process of kernelization in the classical fixed-parameter tractability results.

A problem kernel is an equivalent and reduced problem instance whose size only depends on a given parameter of the original instance. According to a well-known result from classical parameterised complexity literature, a problem is fixed-parameter tractable if and only if there exists a polynomial time reduction to the problem kernel [35]. While some FPT-EAs in the literature have an explicit preprocessing stage where the problem kernel is found, the conventional framework for EAs depends only on iterated variation and selection operators. However, in the case of a few FPT-EAs after a certain number of regular iterations the search process is restricted to a subset of the search space whose size can be bounded by a function of the given parameter k. We will denote this subset of the search space as *the EA-kernel* (Definition 9). The discovery of a solution that belongs to the EA-kernel by an elitist EA ensures that future populations will always have some individuals in the EA-kernel. Since the probability of mutating any solution from the EA-kernel to the optimal solution is bounded from below by a function $1/f(k)$ that only depends on the parameter k, the expected runtime starting from a popu-

lation which consists only of EA-kernel solutions is bounded from above by $\mathcal{O}(f(k))$.

Definition 9. *EA-(FPT/XP)-kernel, S, of a set $\mathcal{X}$ and a function $g : \mathcal{X} \to \mathbb{R}$ such that $\mathcal{X}^* \subseteq \mathcal{X}$ is the optimal subset of $\mathcal{X}$ with respect to g is defined as:*

$$S := \{x \in \mathcal{X} | \forall y \in \mathcal{X}\ g(y) \geq g(x) \implies \Pr(mut(y) \in \mathcal{X}^*) = \Omega(\tfrac{1}{f(k)})\}$$

where for EA-FPT-kernel $f(k)$ is any recursive function that is in the order of $\mathcal{O}(n^c)$ for a $c = \mathcal{O}(1)$ and for any fixed k while for EA-XP-kernel $f(k)$ is any recursive function in the order of $\mathcal{O}(n^{h(k)})$ for some function h that only depends on k.

The EA-kernel is best illustrated in the case of the Planar Euclidean Travelling Salesperson Problem (PETSP) [80]. In polynomial time, the EA finds a solution where the convex hull vertices are in correct order and the offspring solutions which violate that order are not accepted since their objective function values are worse. Even though the number of permutations which are acceptable are bounded by the fixed-parameter (the number of inner vertices) the probability of obtaining any of these permutation requires a correct series of JUMP or INVERSE operations whose individual probabilities depend on n [80]. This dependence on n can be fixed by introducing a pre-processing and/or post-processing stage which allows the EA to work on the real problem kernel, for the case of PETSP, the space of permutations where the cities on the convex-hull appear in a fixed order. This initial stage might be as short as a single iteration if the whole solution space is in the EA-kernel [55]. A similar approach can be seen in the FPT-EA for the VERTEX COVER problem. However, we have to modify our definition of EA-kernel to adapt it to the multi-objective setting by replacing $f(y) \geq f(x)$ with $f(y)$ dominates $f(x)$ [18]. This implies that if a solution in the EA-kernel is found then any solution that dominates it is also a member of the EA-kernel. The multi-objective formulation of the VERTEX COVER problem uses the number of selected vertices as the first objective and the number of uncovered vertices in the remaining graph as the second objective [56]. With this multi-objective formulation, once a solution which is a subset of the optimal vertex cover is found then that solution stays in the population until another solution which is a subset of the optimal vertex cover with less uncovered vertices replaces it.

Hitting the EA-kernel is not a necessity for FPT-EAs. Another approach is to divide the problem into two levels. For instance the FPT-EA

proposed for the MAKESPAN SCHEDULING problem uses restarts after a polynomial length of time in order to search through the assignments of longer jobs and during that polynomial phase optimises the makespan given that the assignment of large jobs is correct. Thus, the exponential waiting time is restricted to the number of restarts until the optimal assignment is found. Similarly, the FPT-EA proposed for PETSP mutates among the permutations of inner points and then uses a deterministic subroutine to find the optimal tour that respects a given permutation of inner points [69]. In this case the runtime which depends exponentially on the number of inner points is restricted to search among the correct permutations.

It is known from the classical fixed-parameter tractability literature that the existence of an FPT-algorithm for a problem implies that the problem can be polynomially reduced to an easier problem whose size is bounded by the fixed-parameter [35]. A similar process is observed in some FPT-EAs where after a few iteration the search process is contained in a subset of the search space (that we name the EA-kernel) where the distance to the optimal solutions is bounded by a particular parameter k of the problem.

3.1.3 Variation operators

The mutation operators used for combinatorial optimization problems are generally tailored to the problem structure. Standard bit mutation (SBM) which independently flips the bits along the string with a certain probability makes the algorithm lose time investigating infeasible solutions. Tailored mutation operators can ensure that an output solution is feasible.

The sufficiently large lower bounds on transition probabilities to the optimal solution x^* is a fundamental obstacle in FPT-EAs. EAs use global operators which always have a non-zero probability of jumping directly to the optimal solution. This probability decreases exponentially by the number of incorrect components, such as bits and edges in the current solution. If an independent change in one component occurs with probability p, a direct jump to the optimal solution, which requires each component to be corrected simultaneously, occurs with probability $p^{\mathcal{O}(n)}$ in the worst case. Correspondingly, the probability that any solution x

which has $f(k)$ incorrect components will be mutated into the optimal solution x is $p^{\mathcal{O}(f(k))}$ If the mutation probability p depends on n (as it is in SBM which flips every bit with probability $p = 1/n$) then we get an XP-EA even if we can restrict the number of incorrect components. This is the reason why many FPT-EAs do not use SBM and prefer tailored mutation operators.

The operator used for MAXIMUM LEAF SPANNING TREE problem does not only ensure feasibility but also works as a key component of fixed-parameter tractability [55] . The reason is that the tree-based operator keeps a tree of $n - 1$ edges and the bound on the number of edges in the graph is of the form $f(k) + n$. Since the tree-based operator picks edges among those that are not already on the tree, the probability of picking a particular edge is bounded from below by $1/f(k)$, an expression which is independent of n.

For the MAKESPAN SCHEDULING problem where the FPT-EA uses restarts to correctly set the first k bits, the mutation operator is modified to have a smaller probability to switch those first k bits [82]. Another example tailored operator is the one used for the VERTEX COVER problem [56]. The multi-objective EA finds a solution which denotes a subset of the optimal vertex cover. The number of components (bits that denote whether a vertex is included in the cover or not) that needs to be changed is then bounded by the size of the minimum cover k. The tailored mutation operator flips a bit with probability $1/2$ if the vertex that corresponds to that bit is covered by the current solution, and with probability $1/n$ otherwise. The probability of mutating a subset of the optimal solution to the optimal solution is therefore $\Omega(1/2^k)$.

Even though it does not play an important role in the final jump to the optimal solution, the mutation operator of the FPT-EA for MAX-2-SAT problem differs from the SBM since the former has a $1/2$ probability of flipping all the bits simultaneously.

The parameterised analysis allowed the theory of EAs to expand further into combinatorial optimisation problems. The initial theoretical results were the adaptation of classical deterministic FPT-algorithms. This adaptation required tailoring of problem-specific mutation operators which can exploit the structure of the problem and prevent the direct effect of input size on mutation probabilities.

3.2 Drift analysis of population-based algorithms

While complex evolutionary algorithms with large populations are common in practice, the initial theoretical results for any algorithm are generally for single-individual versions. This is partly due to the ease of representing a single-individual algorithm as a Markov chain (Chapter 2, Definition 2). Since the progress of any single-individual algorithm over time can be represented as a sequence of positions (a *trajectory*) in the search space, the search process can be represented by a Markov chain which uses the search space of the problem as the state space. The transition probabilities of the Markov chain also follow directly from the definition of the variation operators. On the other hand the underlying Markov Chain of a population-based evolutionary algorithm is a random sequence of points in the population space rather than the search space . The transition probabilities between populations are not explicit either since they are the result of the composition of variation operators on the search space and the selection operators on the population space. This section discusses how drift analysis was used to extend the Markov chain framework from single-individual algorithms to population-based algorithms.

He and Yao introduce drift analysis (Chapter 2, Theorem 6) to runtime analysis of evolutionary algorithms in the early 2000s [43, 46]. The distance function is used to reduce the behaviour of EAs in a higher dimensional population space into a supermartingale on the one-dimensional space [46]. This reduction allows the progress of a simpler stochastic process which moves between different values of the distance function to represent what happens in the population space and allows an approximate Markov chain representation for any algorithm including population-based EAs.

The *drift* of a Markov chain is the expectation of the single-step change in the distance function conditional on the current state. He and Yao provided respective requirements on the expected value of the conditional drift for polynomial and exponential hitting times [43]. The framework is presented to include the population-based algorithms but the distance is defined as a measure on the search space instead of population space. The distance value for a population is defined as the minimum of the distances of individuals in the population. Therefore, populations with

same best individuals are considered as equally distant from finding the optimal solution. Consequently, when bounding the drift from below, the condition on the current distance value of the population provides information for only the best individual and in the worst case bound, the probability of a positive drift carries the probability of selecting that best individual for variation as an overhead multiplier. Since this multiplier is the same regardless of the frequency of individuals with the same or similar distance values, the outcome of the drift analysis underestimates the performance of population-based algorithms.

A crucial step towards finding a more descriptive measure for the state of population-based algorithms was taken by He and Yao who proposed that populations are beneficial for optimising certain functions in polynomial time. While a Markov chain is used to analyse the speedup by the population, the state space is defined as a partition of the population space instead of the search space. The state of the population was defined by two dimensions, the first is the minimal Hamming distance to the optimal solution among all solutions in the population (similarly to [43]), and the second is the number of solutions in the population that share this minimal Hamming distance. By partitioning the population space in this way, the resulting Markov chain is used to analyse the expected hitting-times of a population-based algorithm starting from any of these two dimensional configurations [44] . However, the drift analysis was not in the scope of [44].

A distance function that combines the two dimensional state into a single dimensional state was proposed by He and Yao [46]. This distance function assigns different scalar values for each partition of the population space such that the distance decreases as the smallest Hamming distance to the optimal solution, ℓ, decreases or as number of individuals k with Hamming distance ℓ to the optimal solution increases but only until a certain fraction ($\lceil n/\ell \rceil$) of the population shares the minimum Hamming distance ℓ.

With this distance function two types of transition probabilities are required to bound the conditional drift of the Markov chain. The first is the probability that a population with $k \geq \lceil n/\ell \rceil$ produces an offspring with Hamming distance less than ℓ to the optimal solution. The second is the probability that a population with $k < \lceil n/\ell \rceil$ to replicate a solution with distance ℓ to the optimal solution. When these probabilities are

known and if the population does not accept offspring that have larger Hamming distances to the optimal, the drift can be bounded from below. While this distance function is especially designed for analysing $(N + N)$ EA on ONEMAX, Witt extends the same idea to analyse $(\mu + 1)$ EA on ONEMAX, LEADINGONES and SPC by accounting for the time algorithm spends in increasing the frequency of best solutions in the population though without the mediation of neither the drift analysis nor Markov chain representation in the analysis [86]. Further generalisation of the two dimensional partitioning was presented by Chen et al. in [4] where the time it takes to increase the frequency is framed as the 'takeover time'.

The original purpose of the takeover-time measure is to assess a selection mechanism by how fast the best individuals take over the population if you apply that selection mechanism repetitively without adding any variation to the population [40]. Chen et al. generalized the concept of the takeover time to EAs with mutation and used the generalized γ-takeover time to obtain the mean first hitting time of population- based EAs on unimodal problems [4]. The generalised γ- takeover time maps an initial population to an expected time until the EA obtains a population with a predetermined frequency γ of best solutions. The distance function that consists two additive terms in [4] used γ-takeover as the term which represent the progress in reaching a critical ratio of good solutions The other term is the expected time spent waiting for a better fitness value to be introduced to the population given that the current best fitness has already taken over the population to some degree. This distance function provides a straightforward lower bound of one for the conditional drift and therefore the resulting expected runtime is the sum of these terms over all possible initial populations.

However the results from [4] heavily rely on the elitism of the underlying search process. The adaptation of the takeover time to include processes with mutation in contrast to the definition in [40] which assumes an iterative selection process makes the mutation rate of the process a decisive factor for the takeover time if the selection mechanism is not strictly elitist. Lehre and Yao investigated the relationship between the mutation rate and the selective pressure of selection mechanisms [62]. The selection pressure is defined as a function $\beta : [0, 1] \rightarrow [0, 1]$ where $\beta(\gamma)$ is the probability that the selection mechanism will sample a solu-

tion from among the best γ fraction of the population. Lehre and Yao provided bounds on the selective pressure necessary for a polynomial runtime and also showed exponential lower bounds when the selective pressure is not within the bounds.

Lehre established the general purpose fitness-level theorem for non-elitist population-based algorithms which is independent from the variation operator and selection mechanism, *i.e.* a very large class of algorithms [60]. The theorem provides upper bounds on the runtime of all non-elitist population-based algorithms which sample the individuals in the next generation identically and independently with sufficient selective pressure and population size. Like [4] the distance function used in the analysis has two additive terms, one for the expected takeover time and one for the expected time until an improvement occurs. An asymptotically better bound is achieved with the refined theory in [12] where the term that quantifies the expected takeover time became logarithmic with respect to the current number of good solutions while the expected conditional drift is kept constant. Lehre also presented a general analysis technique which allows one to prove exponential lower bounds on the runtime of non-elitist, population-based EAs [59].

3.3 Evolutionary algorithms for shortest path problems

The shortest path problem is a fundamental combinatorial optimisation problem and still an active field of research which extends to the runtime analysis of evolutionary algorithms. It was among the first combinatorial optimisation problems which were rigorously analysed in the context of evolutionary algorithms [77]. The prior analysis by Scharnow et al. was later ramified by succeeding works to cover different evolutionary algorithms and variations of the shortest path problem.

The complexity of the analysed evolutionary algorithms for the shortest path problem increased progressively. From the algorithmic design perspective, it is typical to use variation and selection operators which are tailored for a specific problem. Still, the antecedent works on runtime analysis of EAs seldom digressed from elementary operators, because the tailored operators complicated the analysis and the runtime results

obtained were problem specific, *i.e.*, did not provide insight for the analysis of other algorithms and problems. Even negative results for elementary algorithms, i.e. that the algorithm requires exponential time to reach a desired solution quality, serve the purpose of pointing out shortcomings of EAs in general. It was not necessary to devise another algorithm to solve an artificial problem. But, when a non-artificial combinatorial optimisation problem like the shortest path problem is analysed and it is proven that the simplest EA takes exponential time to solve it, then the follow up question was "What is the simplest EA that can solve it in polynomial time?". It is one of the first examples of how the analysis of EAs was adapted when faced with complex problems. While the shortest path problem is proven to be exponentially hard to solve for the simplest EA, a chain of works provided positive results by configuring the problem representation, the objective function, the variation and selection operators and by adding auxiliary mechanisms that improve solution quality [3, 22, 25, 23, 27, 26, 47]. Eventually, the requirement for more complex operators made all-pairs shortest path problem the first non-artificial problem where an EA with crossover is proven to perform asymptotically better than an EA without.

Runtime analysis of the shortest path problem in the context of evolutionary algorithms started in 2002 with the work of Scharnow, Tinnefeld and Wegener. They concluded that the SSSP can be solved in $\mathcal{O}(n^3)$ time where $n = |V|$ is the number of vertices [77]. The result is obtained for the multi-objective version of the objective function. The multi-objective function used in this analysis is not to be confused for a real multi-criteria problem where edges have multiple weights. The multi-objectivisation used in this problem is to separate the sum of weights into smaller sums each evaluating the path from the source of a destination vertex. The resulting landscape with $n - 1$ objectives provides the algorithm with the required gradient towards the better solutions even when some of the vertices are not connected to the source in the current solution. These infeasible solutions do not have an objective function value, and constitute a large subset of the search space where an EA can not navigate. Scharnow et al. proved that the single objective version of the problem is not solvable in polynomial time by an EA, as long as infeasible solutions are penalised by an infinitely large weight. Baswana et al. complemented this result later by showing that if the penalty of an infeasible

path is set to $n \cdot w_{max}$, then the $(1+1)$ EA solves the single objective version of the problem in $\mathcal{O}(n^3 \log(n + w_{max}))$ time [3]. The result on the multi-objective function was later improved by Doerr et al. in 2007 and a tight runtime of $\Theta(n^2 \max(\log(n), \ell))$ was established where ℓ is the largest number of edges in a shortest path that connects the source to any vertex [22]. Later a different solution representation and variation operator was introduced by Doerr and Johannsen which is edge-based rather than vertex-based representation, as in previous works. With the help of this representation a multiplicative factor of n^2 in both single objective and multi-objective version was replaced by $m = |E|$, the number of edges in the input graph [25]. With this replacement, the performance in sparse graphs, where $m = o(n^2)$ was proved to be asymptotically better.

The first runtime analysis of an EA on the APSP problem was presented with the emphasis that this problem is the first non-artificial problem where an algorithm with a crossover operator can optimise asymptotically faster than a simple EA without crossover [23]. At the time of this result, genetic algorithms (EAs with crossover operators) were proved to be essential only on artificial pseudo-Boolean problems which were designed for this purpose [50, 61, 51, 52]. In their work on the APSP problem, Doerr et al. proved that the lower bound on the optimisation time of the EA without crossover is in the order of $\Omega(n^4)$ while the EA that uses crossover has an optimisation time of $\mathcal{O}(n^{3.5}\sqrt{\log n})$.

The transition from artificial pseudo-Boolean problems to combinatorial optimisation problem is an important milestone in runtime analysis of evolutionary algorithms. The polynomially solvable shortest path problems were one of the initial stepping stones of this transition.

The rest of this chapter discusses how problem representation, objective algorithms and algorithmic components like mutation, selection, crossover operators evolved in response to the structural properties of the shortest path problems.

The solution representation is trivial for the pseudo-Boolean problems since the bit-string is the intuitive input for both the function and the most common operators of EAs like Standard Bit Mutation. For more complex problems like shortest path where the solution is a directed tree on the input graph, suddenly the alternatives for representing a solution increase. Choosing the correct representation among these alternatives is

important since different solution representations requires different mutation operators and affects the overall performance of the algorithm.

The first representation of the SSSP problem for an EA is a list of $n-1$ predecessor vertices [77]. The i-th entry of the list being u implies that in the shortest path tree the vertex v_i is preceded by the vertex u. An equivalent representation which was proposed later for sparse graphs was to keep a list of edges where the ith entry of the list is an edge with the end vertex v_i [25]. Similar mutation operators were proposed for these list of preceding vertices and list of incoming edges which affect the runtime differently. Both mutation operators pick a random number of local operations to apply, and in each local change pick a position i uniformly at random. The operator for list of predecessor representation randomly picks a new vertex among $V \setminus v_i$ as the predecessor of vertex v_i. The operator for the list of edges, adds a random edge $(u, v) \in E$ at each local operation and removes the current incoming edge of v from the solution. Apparently the second operator never picks a predecessor which is not adjacent to the vertex and this improves the runtime by a factor of $\Theta(n^2/m)$ with respect to the vertex-based representation, a significant factor for sparse graphs.

A list of predecessor vertices or the equivalent list of edges do not necessarily represent a feasible solution because there can be multiple connected components in the resulting graph. The runtime results vary drastically according to how these infeasible solutions are evaluated. An infeasible solution consists of a number of components with cycles and a tree rooted in the source vertex. If the objective function favours solutions with fewer vertices belonging to the components with cycles (off-tree vertices) then the algorithm finds a feasible solution in polynomial time. But if all infeasible solutions are considered equally bad, then it takes exponential time to find a feasible solution [77].

Favouring solutions with fewer off-tree vertices has been accomplished via different mechanisms. The first mechanism is to assign a finite penalty to a solution for every off-tree vertex [3]. In that case the gradual improvement of the solution was possible since "less infeasible" solutions have better objective function value. The second mechanism is to divide the objective function of the SSSP into $n-1$ objectives , each evaluating the shortest path to a particular vertex [77]. In this multi-objective setting the path lengths to the off-tree vertices are considered infinitely

large, but since the algorithm only accepts non-dominated solutions to the population, the solutions with fewer infinite components in their objective function vectors replace the solutions with more infinite components.

The multi-objective approach was possible due to the separability of the general shortest path problem into $|S|$ problems with a single pair of vertices to be connected. Moreover, the multi-objective function turned out to perform better on SSSP [77] [3]. A problem representation which will make use of this separability was proposed by Doerr et al. in their analysis of a genetic algorithm for the all-pairs version of the shortest path problem [23]. In this representation a sequence of vertices which starts with u and ends with v is kept for each pair of vertices $(u, v) \in S$ to be connected. While the tree structure is not necessarily kept, the solutions are always feasible in the sense that all pairs in S are connected.

With $|S|$ sequences, each having a size up to $n - 1$, list of paths representation uses more memory than a predecessor list. Moreover, separating the problem prevents the algorithm from implicitly exploiting one of the most important structural properties of shortest path problems, that subpaths of optimal paths are also optimal. This property could be exploited by finding the optimal paths of smaller number of edges first and then adding edges to these paths to obtain paths with more edges. When the predecessor list representation is used, the EAs implicitly keep an optimal partial solution which gradually grows into a full optimal solution [77] [3] [22]. However, when the paths to different vertices are optimised in parallel as separate problems, exploiting the subpath optimality requires tailored variation operators.

The mutation operator for the list of paths representation consists of a random number of local operations applied to a randomly selected sequence. In each local operation the selected sequence is either shrunk or extended by a single vertex. The new sequence replaces an existing sequence with the same end points and not smaller total weight. By replacing other paths than the one selected for variation, the connection between separated problems are re-established. Similarly, the crossover operator picks two paths randomly and concatenates them. The offspring receives the beginning and end vertices from two different parents and eventually it replaces neither. Like the mutation operator, the offspring replaces the path that shares its endpoints with no smaller ob-

jective function value. These variation operators are indeed tailored to exploit the subpath optimality property. The mutation operator picks a path and then extends it via adding more edges. If the solution already consist of all the optimal paths of i edges then any optimal path of $i+1$ can be obtained by a single correct local operation. For the crossover operator having all the optimal paths of i edges is having optimal building blocks to combine. With the correct choice of parents any path of up to $2i$ edges can be constructed in a single crossover operation.

The crossover operators allows the runtime to be improved asymptotically from $\Theta(n^4)$ for EAs without crossover to $\mathcal{O}(n^{3.5}\log^{1/2}(n))$ [23]. This runtime was proven by first showing that initially the mutation operator finds the shortest paths of smaller number of edges and then showing when such shortest paths of smaller edges are available in the population crossover can produce the longer paths by combining them. An asymptotically tight upper bound of $\mathcal{O}(n^{3.25}\log^{1/4}(n))$ was provided by Doerr and Theile with a corresponding lower bound when these phases are analysed as intersecting phases where crossover initially populates the subset of longer shortest paths and mutation fills the gaps left by the coarse search by crossover [27]. Further work on the ASPS problem introduced changes in the crossover operators either by repairing the offspring solution or by selecting parents in order to ensure feasible solution [26]. With these helpful mechanisms, the runtimes reduced to $\mathcal{O}(n^{3.2}\log^{1/5}(n))$ and $\mathcal{O}(n^3\log(n))$ for the algorithms with repair mechanism and parent selection respectively.

In order to handle more complex combinatorial optimisation problems the field of runtime analysis diverted from its conventional approach of using the simplest possible algorithms. This convention was a by-product of keeping the scope of the field on toy problems whose solutions are only relevant to the practical applications of EAs in terms of excluding bad algorithms and determining reasonable parameter settings. However, when the threshold of relevancy is reached with combinatorial optimisation problems like shortest path problem which constitutes an active research field as it is, the convention of minimalism is abandoned and whenever the simple algorithms had difficulties in solving a problem the algorithmic components from the classical algorithms leaked into the EA piece by piece.

The following chapters will discuss how more involved fitness eval-

uations might be used in EAs solving combinatorial optimisation problems and how the runtime of resulting EAs can be analysed. A fitness function that exploits the structure of bi-level optimisation problems will be used to obtain a fixed-parameter tractability result for an NP-hard optimisation problem. A new mathematical framework that allows the analysis of non-elitist, population-based algorithms that use arbitrary fitness functions for parent selection will be presented. Then, this framework will be used to analyse the runtime of a genetic algorithm for shortest path problems that incorporates components of a well-known deterministic algorithm into its fitness function.

Chapter 4

Fixed-parameter Tractable Evolutionary Algorithms for bi-level optimisation

4.1 Introduction

In recent years, researchers became very interested in bi-level optimisation for single-objective [54, 58] and multi-objective problems [19, 20]. Bi-level problems can be split up into an upper and a lower level problem which depend on each other. By fixing a possible solution for the upper level problem, the lower level is optimised with respect to the given objective and the constraints imposed by the choice of the upper level. Hence, evaluating the objective value of the upper level problem involves solving an optimisation problem itself. The separated structure of the problem allows the use of different approaches to tackle different parts of the problem.

The main motivation behind bi-level optimisation problems are the settings where the decision variables of a problem are set by two agents, which are called the *leader* and the *follower*, with different objectives. Given the decision of the leader, the follower makes a decision according to his objective function which might be conflicting with the objective function of the leader. An example of such a problem is where the leader places toll booths across a road network and the followers try to find the cheapest way from a point A to a point B by finding a path that avoids as many toll booths as possible. Here, the leader can only learn the objective function value of its decision after the followers pick the optimum

path.

We will start our analyses of complex function evaluation methods with the runtime analyses of two bi-level combinatorial optimisation problems. The problems we will examine in this work are bi-level optimisation problems where the upper level problem and the lower level problem share an objective function. Unlike problems with conflicting upper and lower level objective functions, it is not a necessity to model our problems as bi-level optimisation problems. We will use the bi-level structure in order to divide the problem so that we can delegate the lower level problem to a deterministic algorithm.

Recently, Hu and Raidl [49, 48] have proposed two different approaches for the generalised minimum spanning tree problem (GMSTP). Both approaches work with an upper level and a lower level solution. The upper level solution x is evolved by an evolutionary algorithm whereas the optimal solution y of the lower level problem corresponding to a particular search point x of the upper level can be found in polynomial time using deterministic algorithms. From the perspective of the evolutionary algorithm which searches through upper level solutions, the deterministic algorithms which optimise the lower level problem are function evaluations.

In order to understand and compare the two different approaches presented by Hu and Raidl [49, 48], we push forward the parameterised analysis of evolutionary algorithms (see Section 3.1) and present the first analysis in the context of bi-level optimisation. In our investigations, we consider the two NP-hard problems the generalised minimum spanning tree problem (GMSTP) and the generalised travelling salesman problem (GTSP) which share the number of clusters m as parameter. We consider two different bi-level representations for GMTSP which both have a polynomially solvable lower level part. We use problem-specific polynomial deterministic algorithms to solve the lower level parts and restrict the search of the evolutionary algorithm to the upper level problem. For the *Spanned Nodes Representation*, we present worst case examples which show that there are instances leading to a runtime of $\Omega(n^m)$. For the *Global Structure Representation*, we show that it leads to a fixed-parameter evolutionary algorithm with respect to the number of clusters m. Furthermore, we present an instance class where the algorithm using the *Global Structure Representation* encounters an runtime of $m^{\Omega(m)}$.

Analysing both approaches on each others worst-case instances, we show that they solve them very efficiently. This shows the complementary abilities of these two representations for the GMSTP. Then we extend our results for *Global Structure Representation* to GTSP to show that a similar algorithm has an expected runtime of $m^{\Omega(m)}$ for this problem as well.

The chapter is divided into two main parts according to the two different problems. The first part (based on the conference version [9]) where the GMSTP problem is investigated is presented in Section 4.2. We show hard instances for the *Spanned Nodes Representation* in Section 4.2.1 and show that a simple evolutionary algorithms needs exponential time even if the number of clusters is small. In Section 4.2.2, we examine the *Global Structure Representation* and show that this leads to fixed-parameter evolutionary algorithms for GMSTP. We point out complementary abilities of the representations in Section 4.2.3. In the second part (based on journal article [10]), we examine the GTSP problem with the corresponding *Global Structure Representation* in Section 4.3 and provide upper and lower bounds on the runtime of the considered algorithm. Furthermore, we point out in Section 4.4 general characteristics which allow this fixed-parameter result to be extended to other problems.

4.2 Generalised Minimum Spanning Tree Problem

In this section, we consider the GMSTP (Section 2.3.1) and analyse the runtime with respect to the bi-level representations given in [49, 48].

The GMSTP problem is NP-hard [68], hence two different bi-level evolutionary approaches were examined in [48]. The first approach presented in [48] uses the *Spanned Nodes Representation*. It selects in the upper level problem a node for each cluster and computes on the lower level a minimum spanning tree (using for example Kruskal's algorithm in time $O(m \log m)$ on the induced subgraph).

4.2.1 Spanned Nodes Representation

Algorithm 4 Cluster-based (1+1) EA

Choose a random spanned nodes set $P \in V^m$.
Let T be a MST of $G[P]$.
while termination condition not satisfied **do**
  $P' \leftarrow P$
  for $i \in [m]$ **do**
    with probability $1/m$, sample $p'_i \sim \mathrm{Unif}(V_i)$.
  end for
  Let T' be the MST of $G[P']$.
  if $C(T') \leq C(T)$ **then**
    $P \leftarrow P'$
    $T \leftarrow T'$
  end if
end while

We analyse the cluster-based (1+1) EA (Algorithm 4) in this section. Our first theorem shows that this algorithm is an XP-algorithm, i.e. an algorithm that runs in time $O(n^{g(m)})$ where $g(m)$ is a computable function only depending on m, when choosing the number of clusters m as a parameter.

Theorem 10. *For any instance of the GMSTP problem, the expected time until the cluster-based (1+1) EA reaches the optimal solution is $O(n^m)$.*

Proof. For any search point x, let $w(x) \in [m]$ denote the number of clusters where the spanned node representation includes a suboptimal node. If the algorithm chooses all $w(x)$ suboptimal clusters for mutation and selects the optimal node in each of them, then the optimal solution is obtained. Since $w(x) \leq m$, the probability that all suboptimal clusters are mutated in a single step is at least $(1/m)^m$. The probability of choosing the optimal node in cluster i is $1/|V_i|$. Thus, the probability of jumping to the optimal solution from any search point is at least

$$(m)^{-m} \prod_{i=1}^{m} |V_i|^{-1}.$$

Since $\sum_{i=1}^{m} |V_i| = n$, it holds that

$$\prod_{i=1}^{m} \frac{1}{|V_i|} \geq (m/n)^m.$$

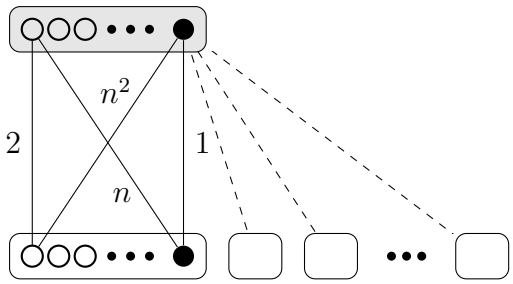

Figure 4.1: Hard instance G_S for *Spanned Nodes Representation*.

Therefore, the probability of reaching the optimal solution in one step is $\Omega(n^{-m})$, and the expected time to reach the optimal solution is bounded from above by $O(n^m)$. □

We now consider an instance of GMSTP which is difficult for the cluster-based (1+1) EA in order to bound the runtime from below. The hard instance G_S for the *Spanned Nodes Representation* is illustrated in Figure 4.1. It consists of m clusters, where one cluster is called the *central* cluster, and the $m-1$ other clusters are called *peripheral* clusters. We assume that $n \geq m^2$ holds and that n is divisible by m. Each cluster contains n/m nodes. The nodes in the peripheral clusters are called peripheral nodes, and the nodes in the central cluster are called central nodes. Within each cluster, one of the nodes is called *optimal*, and is marked black in the figure. The remaining $(n/m)-1$ nodes are called *sub-optimal* nodes, and are marked white in the figure. The instance is a bi-partite graph, where edges connect peripheral nodes to central nodes. The cost of any edge between two optimal nodes is 1, the cost of any edge between two suboptimal nodes is 2. The cost of any edge between a suboptimal peripheral node and the optimal central node is n^2, and the cost of any edge between an optimal peripheral node and a suboptimal central node is n. A cluster is called *optimal* in a solution, if the solution has chosen the optimal node in that cluster.

Theorem 11. *Starting with an initial solution chosen uniformly at random, the expected runtime of the cluster-based (1+1) EA on G_S is $\Omega(n^m)$. Furthermore, for any constant $\epsilon > 0$, the probability of having obtained an optimal solution after at most $n^{(1-\epsilon)m}$ iterations is $e^{-\Omega(m)}$.*

Proof. We define two phases for the run of the (1+1) EA. The first phase consists of the first $n-1$ iterations while the second phase starts at the end of the first phase and continues for $n^{m/12}$ iterations. Four distinct

events are considered failures during the run of the (1+1) EA for the instance described above.

1. The first failure occurs if during the first phase of the run, the algorithm obtains a search point with less than $m/6$ sub-optimal peripheral clusters.

2. The second type of failure occurs when the central cluster fails to switch to a suboptimal node at least once during the first phase.

3. The third failure corresponds to a direct jump to the optimal solution during the second phase.

4. The fourth type of failure occurs when the algorithm does not switch all the optimal peripheral clusters to suboptimal clusters during the second phase.

We first show that the probability of the first failure event is at most $\exp(-\frac{m}{12})$. This implies that with overwhelmingly high probability, a constant fraction of peripheral clusters is always suboptimal during the first $n-1$ iterations. For $i \in [m-1]$ and $t \geq 0$, let $Z_i(t)$ be a random variable such that $Z_i(t) = 1$ if cluster V_i is always sub-optimal in iteration 0 through iteration t, and $Z_i(t) = 0$ otherwise. The probability that a suboptimal node is selected in the initial solution is $1-m/n$. In the following iterations, the probability that a cluster is selected for mutation and that its new spanned node is optimal is $(1/m)(m/n) = 1/n$. So it is clear that

$$\Pr\left(Z_i(t) = 1\right) \geq (1 - m/n)(1 - 1/n)^t.$$

By linearity of expectation,

$$E\left[\sum_{i=1}^{m-1} Z_i(t)\right] \geq (m-1)\left(1 - \frac{m}{n}\right)\left(1 - \frac{1}{n}\right)^t.$$

Considering a phase length of $t = n - 1$, and assuming that m is sufficiently large and $n \geq m^2$ holds, we get

$$E\left[\sum_{i=1}^{m-1} Z_i(t)\right] \geq \frac{m}{3}.$$

Finally, a Chernoff bound (Theorem 8) implies that

$$\Pr\left(\sum_{i=1}^{m-1} Z_i(t) \leq \left(1-\frac{1}{2}\right)\frac{m}{3}\right) \leq \exp\left(-m/12\right).$$

We then show that the probability of the second failure event is $\exp(-\Omega(\sqrt{n}))$. In each iteration the probability to switch the central cluster to a suboptimal node is at least

$$p = \frac{1}{m}\left(1-\frac{m}{n}\right) = \Omega\left(\frac{1}{\sqrt{n}}\right).$$

The probability that this event does not occur in $n-1$ steps is

$$\begin{aligned}(1-p)^{n-1} &= \left((1-p)^{1/p}\right)^{(n-1)p} \\ &\leq \exp(-p(n-1)) = \exp(-\Omega(\sqrt{n})).\end{aligned}$$

Now, we show that the probability of the third failure event is less than $n^{-m/12}$, assuming that the first two failure events do not occur. As long as the central cluster remains suboptimal, switching a suboptimal node in a peripheral cluster to an optimal node will result in an extra cost of $n-2$. Conversely, switching an optimal peripheral cluster into a sub-optimal cluster will decrease the cost by $n-2$. As long as there is at least one suboptimal peripheral cluster, making the central cluster optimal will incur an extra cost of at least n^2-2. So, during phase two, the algorithm can not make any suboptimal cluster optimal unless all suboptimal clusters are made optimal in the same iteration. The probability of making at least $m/6$ suboptimal peripheral clusters optimal simultaneously is at most

$$\left(\frac{1}{m}\cdot\frac{m}{n}\right)^{m/6} = \left(\frac{1}{n}\right)^{m/6}.$$

Since the probability to jump to the optimal solution is at most $n^{-m/6}$ in each iteration, it holds by a union bound that the probability of failure event three is at most

$$n^{-m/6}n^{m/12} = n^{-m/12}.$$

Finally, we show that the probability of failure event four is $O(n^{-m/12+4})$. The probability that an optimal peripheral cluster is made suboptimal by

the (1+1) EA is at least

$$\frac{1}{m} \cdot \frac{n-m}{n} \cdot \left(1 - \frac{1}{m}\right)^{m-1} \geq \frac{1}{3m}.$$

The expected time $E[T^{sub}]$ until all peripheral clusters have become sub-optimal is therefore at most $m \cdot 3m = O(m^2)$. Considering a phase of length $n^{m/12}$ and taking into account $n \geq m^2$, it holds by Markov's inequality that the probability of a type four failure is

$$\Pr\left(T^{sub} > n^{m/12}\right) \leq O(m^2)n^{-m/12} = O(n^{-m/12+4}).$$

By a union bound, the probability that any type of failure occurs is less than the sum of their independent probabilities, which is $e^{-\Omega(m)}$. Hence, with overwhelmingly high probability, after the second phase, the algorithm has obtained a locally optimal solution where all peripheral clusters are sub-optimal. After that iteration, the probability to jump directly to the optimal solution is n^{-m}, and the expected time for this event to occur is n^m.

Let E be the event that no failure occurs. Then, the first statement of the theorem follows by the law of the total probability,

$$\begin{aligned} E[T] &\geq E[T|E] \Pr(E) \\ &= \Omega(n^m)(1 - e^{-\Omega(m)}) \\ &= \Omega(n^m). \end{aligned}$$

Furthermore, by union bound, it holds that

$$\Pr\left(T < n^{(1-\epsilon)m} \mid E\right) \leq n^{(1-\epsilon)m}n^{-m} = n^{-\epsilon m}.$$

Hence, the second statement of the theorem follows by the law of total probability

$$\begin{aligned} \Pr\left(T < n^{(1-\epsilon)m}\right) &= \Pr\left(T < n^{(1-\epsilon)m} \mid E\right) \Pr(E) \\ &\quad + \Pr\left(T < n^{(1-\epsilon)m} \mid \overline{E}\right) \Pr\left(\overline{E}\right) \\ &\leq \Pr\left(T < n^{(1-\epsilon)m} \mid E\right) + \Pr\left(\overline{E}\right) \\ &\leq n^{-\epsilon m} + e^{-\Omega(m)} = e^{-\Omega(m)}. \end{aligned}$$

□

Our results for the *Spanned Nodes Representation* show that the cluster-based (1+1) EA obtains an optimal solution in time $O(n^m)$ and our analysis for the hard instance G_S shows that this bound is tight.

4.2.2 Global Structure Representation

The second approach we will examine is the *Global Structure Representation* [48]. It works on the complete graph $H = (V', E')$ obtained from the input graph $G = (V, E, c)$. The node $v_i \in V'$, $1 \leq i \leq m$, represents the cluster V_i of G.

The upper level solution in the *Global Structure Representation* is a spanning tree T of H and the lower level solution is a set of nodes $P = (p_1, \ldots, p_m)$ with $p_i \in V_i$ that minimises the cost of a spanning tree which connects the clusters in the same way as T. Given a spanning tree T of H, the set of nodes P can be computed in time $O(n^2)$ using the dynamic programming approach of Pop [75].

We consider the tree-based (1+1) EA outlined in Algorithm 5. It starts with a spanning tree T of H that is chosen uniformly at random. In each iteration, a new solution T' of the upper level is obtained by performing K edge-swaps to T. Here the parameter K is chosen according to the Poisson distribution with expectation 1. In one edge swap, an edge e currently not present in the solution is introduced and an edge from the resulting cycle is removed such that a new spanning tree of H is obtained. After having produced the offspring T', the corresponding set of nodes P' is computed using dynamic programming. P and T are replaced by P' and T' if the cost of the new solution is not worse than the cost of the old one.

In the following, we show that the tree-based (1+1) EA is a fixed-parameter evolutionary algorithm for the GMSTP problem when considering the number of clusters m as the parameter. We do this by transferring the result for the runtime of the dynamic programming approach of Pop [75] to the tree-based (1+1) EA.

Theorem 12. *The expected time of the tree-based (1+1) EA to find the optimal solution for any instance of the GMSTP problem is $O(m^{3(m-1)})$. Furthermore, for any $k \geq 1$, the probability that an optimal solution is not found within $ekm^{3(m-1)}$ steps is less than $\exp(-k)$.*

Proof. An upper level solution is a tree T of H. Let T^* be any tree of H for

Algorithm 5 Tree-based (1+1) EA

Choose a spanning tree T of H.
Apply dynamic programming to find the minimum spanned nodes $P = (p_1, \ldots, p_m)$ induced by T.
while termination condition not satisfied **do**
  $T' \leftarrow T$
  for $i \in [K]$ where $K \sim \text{Pois}(1)$ **do**
    Sample edge $e \sim \text{Unif}\,(E' \setminus T)$
    Sample edge $e' \sim \text{Unif}$(edges in cycle in $T' \cup \{e\}$)
    $T' \leftarrow T' \cup \{e\} \setminus \{e'\}$
  end for
  Apply dynamic programming to find a set of spanned nodes $P' = (p'_1, \ldots, p'_m)$ with respect to T' of minimum cost.
  if $\sum_{(i,j)\in T'} c(p'_i, p'_j) \leq \sum_{(i,j)\in T} c(p_i, p_j)$ **then**
    $P \leftarrow P'$
    $T \leftarrow T'$
  end if
end while

which there exists a set P^* of spanned nodes such that T^* and P^* form an optimal solution. For any non-optimal solution T, define $w(T)$ as the number of edges in T^* which are missing in T.

The mutation operator can convert a non-optimal solution T into an optimal solution T^* with a sequence of $w(T) \leq m-1$ edge exchange operations. The probability that the mutation operator exchanges $w(t) \leq m-1$ edges in one mutation step is at least

$$\Pr\left(\text{Pois}(1) = m-1\right) = 1/e(m-1)!.$$

In each exchange operation, if there are i optimal edges missing, then the probability that one of the missing optimal edges is inserted is at least i/m^2. After the addition of an optimal edge, the probability of excluding a non-optimal edge is at least $1/m$ since the largest cycle cannot be longer than m. At most $m-1$ non-optimal edges must be exchanged in this manner. So the probability that the non-optimal solution T will be converted to an optimal solution T^* in one mutation step is at least

$$\frac{1}{e(m-1)!} \cdot \prod_{i=1}^{m-1} \frac{i}{m^2} \cdot \frac{1}{m} = (1/e)m^{-3(m-1)}.$$

So, the expected time to achieve an optimal solution is in $O(m^{3(m-1)})$. Furthermore, the probability that an optimal solution has not been cre-

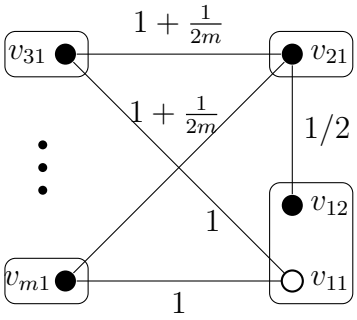

Figure 4.2: Hard instance G_G for *Global Structure Representation*. Edges not shown have weight ∞.

ated after $ekm^{3(m-1)}$ iterations is

$$(1 - (1/e)m^{-3(m-1)})^{ekm^{3(m-1)}} \leq \exp(-k).$$

□

We now present an instance which is hard for the tree-based (1+1) EA. The instance G_G, illustrated in Figure 4.2, consists of n nodes and m clusters. There are two central clusters denoted by V_1 and V_2. The cluster V_1 contains the two nodes v_{11} and v_{12}. The remaining clusters $V_i, 2 \leq i \leq m$, contain a single node v_{i1} each. The edges that connect the nodes v_{11} to the peripheral cluster nodes have cost 1. The edges that connect v_{21} to the peripheral clusters have weight $1 + 1/2m$. The edge that connects v_{12} and v_{21} have weight $1/2$. All other edges have cost ∞. Hence, if the tree-based (1+1) EA connects cluster V_1 and V_2, then the dynamic programming algorithm will choose node v_{12}.

In our analysis, we will use the following lemma on basic properties of the Poisson distribution with expectation 1.

Lemma 13. *If* $K \sim \text{Pois}(1)$, *then* $\Pr(K \geq n) < 2(e/n)^n$.

Proof. Using Stirling's approximation of the factorial,

$$n! > \sqrt{2\pi n}(n/e)^n > (n/e)^n.$$

we obtain the simple bound

$$\begin{aligned}
\Pr(K \geq n) &= \sum_{i=n}^{\infty} \frac{1}{ei!} \\
&< \sum_{i=n}^{\infty} \frac{1}{i!} \\
&< \sum_{i=n}^{\infty} \frac{1}{n!} \left(\frac{1}{n+1}\right)^{i-n} \\
&< (e/n)^n \sum_{i=0}^{\infty} \left(\frac{1}{n+1}\right)^{i} \\
&= (e/n)^n \left(1 + \frac{1}{n}\right).
\end{aligned}$$

□

Using the previous lemma, we are able to show that the tree-based (1+1) EA finds it hard to optimise G_G when choosing a spanning tree uniformly at random among all spanning trees having weight less then ∞.

Theorem 14. *Starting with a spanning tree chosen uniformly at random among all spanning trees that have cost less than ∞, the expected optimisation time of the tree-based (1+1) EA on G_G is $\Omega((m/e)^{m-1})$.*

Proof. Consider the instance in Figure 4.2. In the following, edge $e := \{v_{12}, v_{21}\}$ is the edge which connects the two central clusters. The optimal solution corresponds to the spanning tree which includes edge e, and where all other clusters are connected to cluster V_2. The solution where all peripheral clusters are connected to V_1, and where cluster V_2 is connected to one of the peripheral clusters, is a local optimum.

We define four failure events that can occur during a run of the (1+1) EA on this instance.

1. The first type of failure occurs when the initial solution includes edge e.

2. The second type of failure occurs when less than $m/3$ of the peripheral clusters are connected to cluster V_1 in the initial solution.

3. The third type of failure occurs when the algorithm jumps directly to the optimal solution during the first $((m-2)/3e)^{(m-2)/6}$ iterations.

4. Finally, the fourth type of failure occurs if after iteration $((m-2)/3e)^{(m-2)/6}$, there exists a peripheral cluster which is not connected to cluster V_1.

There are $m-2$ peripheral clusters which must be connected to either V_1 or V_2. Additionally, cluster V_1 and V_2 must be connected. This connection can be established either by adding edge $e = (v_{12}, v_{21})$, or by connecting a peripheral cluster to both V_1 and V_2. There are 2^{m-2} spanning trees which contain edge e, and $(m-2) \cdot 2^{m-3}$ spanning trees which do not contain edge e since one of the $m-2$ peripheral clusters will be connected to both central clusters and the others will be connected to only one. So, the probability that a uniformly chosen spanning tree includes edge e is $O(1/m)$, which is the probability of the first type of failure.

Now, we show that the probability of the second type of failure is at most $\exp(-\Omega(m))$. Considering that the probability of a specific cluster is adjacent to V_1 in the initial solution is larger than $1/2$, the probability that less than $(m-2)/3$ clusters are connected to cluster V_1 in the initial solution is bounded by $\exp(-\Omega(m))$ using a Chernoff bound.

Assuming that type one and type two failures did not occur, the algorithm cannot accept new search points where a cluster which is originally connected to V_1 is instead connected to V_2 since it will create an extra cost of $1/2m$. The only exception is if a type three failure occurs, i.e. the algorithm jumps directly to the optimal solution where all the peripheral clusters are connected to V_2. For a type three failure to occur, at least $(m-2)/3$ clusters have to be modified simultaneously. Therefore, using Lemma 13, the probability of jumping directly to the optimal solution in a single step is bounded from above by

$$2(3e/(m-2))^{(m-2)/3}.$$

Taking a phase length of $((m-2)/3e)^{(m-2)/6}$ into account, the probability of a type three failure can be bounded from above using a union bound, as

$$((m-2)/3e)^{(m-2)/6} 2(3e/(m-2))^{(m-2)/3} = (m/e)^{-\Omega(m)}.$$

Now, it will be shown that the probability of a type four failure is $e^{-\Omega(m)}$. The probability that a single peripheral cluster which is con-

nected to V_2 is switched to V_1 is bounded from below by

$$\frac{1}{3} \cdot \frac{1}{e} \cdot \frac{1}{(m^2 - (m-1))} = \Omega(1/m^2).$$

Thus, the expected time between any such event is $O(m^2)$, and the expected time until all of the at most $m-2$ peripheral clusters are connected to V_1 is $E[T'] = O(m^3)$. Now, we can prove our claim about the probability of failure event four. By Markov's inequality, it holds for any nonnegative random variable X that

$$\Pr(X \geq k) \leq \frac{E[X]}{k}.$$

The probability that it takes longer than

$$k = ((m-2)/(3e))^{(m-2)/6}$$

iterations is therefore no more than

$$\frac{E[T']}{k} = O(m^3) \cdot ((m-2)/(3e))^{-(m-2)/6} = (m/e)^{-\Omega(m)}.$$

If none of the above mentioned failures occur, we reach the local optimum where all the peripheral clusters are connected to cluster V_1. From this point on, the probability to jump to the optimal solution is by Lemma 13 no more than

$$2(e/(m-1))^{m-1}$$

because it is necessary to make at least $m-1$ edge exchanges to reach the optimum. The expected time to reach the optimal solution conditional on no failure is therefore more than $(1/2)(m/e)^{m-1}$.

Let R be the event that no failure occurs. By the law of total probability, it follows that the expected time $E[T]$ to reach the global optimum is

$$\begin{aligned} E[T] &\geq E[T|R] \Pr(R) \\ &= \Omega((m/e)^{m-1})(1 - O(1/m)) \\ &= \Omega((m/e)^{m-1}). \end{aligned}$$

□

The previous theorem shows that there are instances for the cluster-based (1+1) EA where the runtime grows exponentially with the number of clusters. In the next section, we will compare the two different representations for GMSTP and show that they have complementary capabilities.

4.2.3 Complementary Abilities

The two representations examined in the previous sections differ significantly from each other. They both rely on the fact that there is a deterministic algorithm which solves the lower level problem in polynomial time. In this section, we want to examine the differences between the two approaches. We show that both representations have complementary abilities and do this by examining the algorithms on each others hard instance. Surprisingly, we find out that the hard instance for one algorithm becomes easy to solve when giving it as an input to the other algorithm.

In Section 4.2.1, we have shown a lower bound of $\Omega(n^m)$ for the cluster-based (1+1) EA using the *Spanned Node Representation*. The hard instance G_S for the cluster-based (1+1) EA given in Figure 4.1 consists of a central cluster to which all the other clusters are connected. There are no other connections between the clusters. Hence, there is only one spanning tree when working with the *Global Structure Representation*. The dynamic programming algorithm that runs on the lower level of the tree-based (1+1) EA therefore solves the problem in its first iteration.

The following theorem shows that these instances are easy to optimise with the tree-based (1+1) EA.

Theorem 15. *The tree-based (1+1) EA solves the instance G_S in expected constant time.*

Proof. There is only a single tree over the cluster graph. Hence, the algorithm selects the optimal tree in the initial iteration. □

For the tree-based (1+1) EA, working with the *Global Structure Representation*, we showed that it finds the instance G_G given in Figure 4.2 hard to solve. Working with the *Spanned Nodes Representation*, there is only one cluster that consists of two nodes where all the other clusters

contain exactly one node. Hence, an optimal solution is obtained by computing a minimum spanning tree on the lower level if the right node in the cluster of two nodes is chosen. The following theorem summarises this and shows that this instance becomes easy when working with the cluster-based (1+1) EA.

Theorem 16. *The cluster-based (1+1) EA solves the instance G_G in expected time $O(m)$.*

Proof. Cluster V_1 contains two nodes, and all other clusters contain a single node. If the initial solution is not already the optimal solution, then the correct node of V_1 has to be selected using mutation. The node for the cluster V_1 is changed with probability $1/m$ and in such a step the correct node is selected with probability $1/2$. Hence, the probability of a mutation leading to an optimal solution is at least $1/(2m)$ and the expected waiting time for this event is $O(m)$. □

The investigations in this section show that the two examined representations have complementary abilities. Switching from one representation to the other can significantly reduce the runtime.

4.3 Generalised Travelling Salesman Problem

We now turn our attention to the NP-hard generalized travelling salesperson problem (GTSP).

The bi-level approach that we are studying is similar to the one discussed in the previous section. We investigate the *Global Structure Representation* which works on the complete graph $H = (V', E')$ obtained from the input graph $G = (V, E, c)$. The node $v_i \in V'$, $1 \leq i \leq m$, represents the cluster V_i of G.

The upper level solution in the *Global Structure Representation* is a Hamiltonian tour π on H and the lower level solution is a set of nodes $P = (p_1, \ldots, p_m)$ with $p_i \in V_i$ that minimises the cost of a Hamiltonian tour which connects the clusters in the same way as π. Given the restriction imposed by the Hamiltonian tour π of H, finding the optimal set of nodes P can be done in time $O(n^3)$ by using any shortest path algorithm. One such algorithm is *Cluster Optimisation* proposed initially by Fischetti *et al.* [34] which is widely used in the literature. Let $\pi = (\pi_1, \ldots, \pi_m)$ be a

Algorithm 6 Tour-based (1+1) EA

1: Choose a random permutation π (which is also a Hamiltonian tour) of the m given clusters.
2: Find the set of nodes P (one node in each cluster) to build the shortest path possible among those clusters with the given order, by means of any shortest path algorithm in time $O(n^3)$.
3: **while** termination condition not satisfied **do**
4: $\pi' \leftarrow \pi$
5: **for** $i \in [K]$ where $K \sim 1 + Pois(1)$ **do**
6: Choose two nodes from π' uniformly at random.
7: $\pi' \leftarrow$ Perform the *Jump* with the chosen nodes on π'
8: **end for**
9: Find the set of nodes $P' = (p'_1, \ldots, p'_m)$ which minimizes the cost with respect to π' in the lower level
10: **if** $c(\pi') \leq c(\pi)$ **then**
11: $P \leftarrow P'$
12: $\pi \leftarrow \pi'$
13: **end if**
14: **end while**

permutation on the m clusters and p_i be the chosen node for cluster V_{π_i}, $1 \leq i \leq m$. Then the cost of the tour π is given by

$$c(\pi) = c(p_m, p_1) + \sum_{i=1}^{m-1} c(p_i, p_{i+1}).$$

Our proposed algorithm starts with a random permutation of clusters which is always a Hamiltonian tour π, in a complete graph H. In each iteration, a new solution π' of the upper level is obtained by the commonly used *Jump* operator which picks a node and moves it to a random position in the permutation. The number of jump operations carried out in a mutation step is chosen according to $1 + Pois(1)$, where $Pois(1)$ denotes the Poisson distribution with expectation 1. Although we are using the jump operator in these investigations, similar results can be obtained for other popular mutation operators such as *exchange* and *inversion*.

Theorem 17. *The expected runtime of the tour-based (1+1) EA is* $O(m!m^{2m-1})$.

Proof. We consider the probability of obtaining the optimal tour π^* on the global graph H from an arbitrary tour π. The number of *Jump* operations required is at most m (the number of clusters). The probability of picking the right node and moving it to the right position in each of those m

operations is at least $1/m^2$. We can obtain an optimal solution by carrying out a sequence of m jump operations where the ith operation jumps element π_i^* in π to position i. Since the probability that $Pois(1)+1 = m$ is $1/(e(m-1)!)$, the probability of a specific sequence of m *Jump* operations to occur is bounded from below by

$$\frac{1}{e(m-1)!} \cdot \frac{1}{m^{2m}}.$$

Therefore, the expected waiting time for such a mutation is

$$\left(\frac{1}{e(m-1)!} \cdot \frac{1}{m^{2m}}\right)^{-1} = O(m!m^{2m-1})$$

which proves the upper bound on the expected runtime. □

Note that this upper bound depends on the number of clusters. Since the computational effort required to assess the lower level problem is polynomial in the input size $O(n^3)$, the proposed algorithm is a fixed-parameter evolutionary algorithm for the GTSP problem with respect to the parameter m, the number of clusters.

So far we have found an upper bound for the expected time of finding an optimal solution using the presented algorithm. In this section we will find a lower bound for the runtime. Figure 4.3 illustrates an instance of GTSP, G_G, for which finding the optimal solution is difficult by means of the presented bi-level evolutionary algorithm with Global Structure Representation. In this graph, each cluster has two nodes. On the upper level a tour for clusters is found by the EA and on the lower level the best node for that tour is found within each cluster. All white nodes (which represent sub-optimal nodes) are connected to each other, making any permutation of clusters a Hamiltonian tour even if the black nodes are not used. All such connections have a weight of 1, except for those which are shown in the figure which have a weight of 2. All edges between a black node and a white node and also all edges between black nodes have weight m^2, except the ones presented in the figure which have weight $1/m$. An optimal solution of cost 1 uses only edges of cost $1/m$ whereas local optimal solutions use only edges of cost 1. The tour comprising all black nodes in the same order as illustrated in Figure 4.3 is the optimal solution. Note that there are many local optimal solutions of cost m. For our analysis it is just important that they do not share any

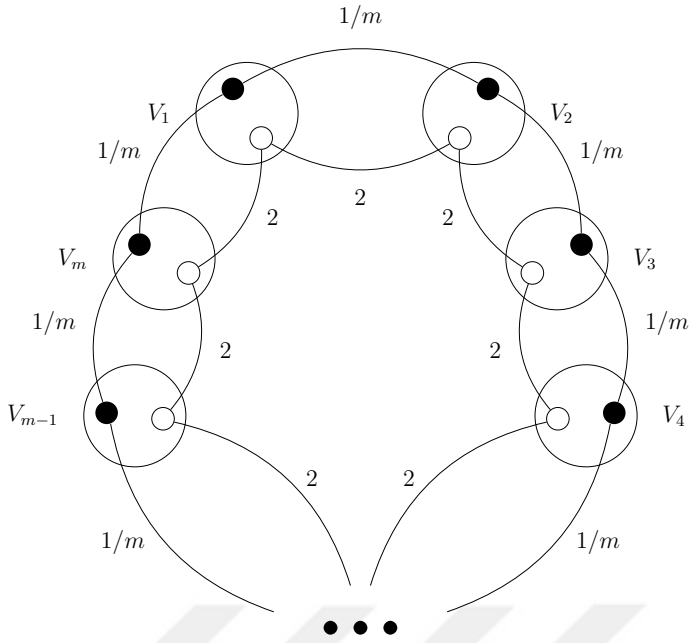

Figure 4.3: A hard instance G_G for GTSP with *global structure representation*

edge with an optimal solution.

The clusters are numbered in the figure, and a measure S for evaluating cluster orders is based on this numbering: Let $\pi = (\pi_1, \ldots, \pi_m)$ represent the permutation of clusters in the upper level, then $S(\pi) = |\{i \mid \pi_{(i+1 \mod m)} = (\pi_i + 1) \mod m\}|$ indicates the similarity of the permutation with the optimal permutation. A large value of $S(\pi)$ means that many clusters in π are in the same order as in the optimal solution. Note that $S(\pi^*) = m$ for an optimal solution π^*. A solution π with $S(\pi) = 0$ is locally optimal in the sense that there is no strictly better solution in the neighbourhood induced by the jump operator. The solutions with $S(\pi) = 0$ form a plateau where all solutions differ from the optimal solution by m edges.

We first introduce a lemma that will later help us with the proof of the lower bound on the runtime.

Lemma 18. *Let π and π' be two non-optimal cluster permutations for the instance G_G. If $S(\pi') > S(\pi)$ then $c(\pi') > c(\pi)$.*

Proof. In the given instance, all white nodes are connected to each other with a maximum weight of 2. These connections ensure that any permutation of the clusters can result in a Hamiltonian tour with a cost of at most $2m$. Moreover, all connections between white nodes and black

nodes have a weight of m^2. So the lower level algorithm will never choose a combination of white and black nodes because the cost will be more than m^2 while there is an option of selecting all white nodes with the cost of at most $2m$. On the other hand, for any permutation of clusters other than the Global Optimum, the lower level will not choose any black nodes, because it will not be possible to use all the $1/m$ edges and some m^2-weighted edges will be used again. Let $a = S(\pi)$ be the number of clusters adjacent to each other correctly from the right side (having the same right-side neighbour as in the Global Optimum) in a solution π. Then $b = m - a$ is the number of clusters which have a different neighbour on their right. If π is not the optimal solution, then the lower level will choose all white nodes. As a result, a edges with weight 2 and b edges with weight 1 will be used in that solution; therefore, the total cost of solution π will be $c(\pi) = 2a + b = 2a + m - a = m + a$. Consider a solution π' with $a' = S(\pi')$ and $S(\pi') > S(\pi)$. We have $c(\pi') = m + a' > m + a = c(\pi)$ which completes the proof. □

Lemma 18 shows that any non-optimal offspring π' of a solution π is not accepted if it is closer to an optimal solution π^*. This means that the algorithm finds it hard to obtain an optimal solution for G_G and leads to an exponential lower bound on the runtime as shown in the following theorem.

Theorem 19. *Starting with a permutation of clusters chosen uniformly at random, the runtime of the tour-based (1+1) EA on G_G is $\Omega((\frac{m}{2})^{\frac{m}{2}})$ with probability $1 - e^{-\Omega(m)}$.*

Proof. Considering G_G illustrated in Figure 4.3, the optimal solution is the tour comprising all edges with weight $\frac{1}{m}$. We consider a typical run of the algorithm consisting of a phase of $T = Cm^3$ steps where C is an appropriate constant. For the typical run we show the following:

1. A local optimum π with $S(\pi) = 0$ is reached with probability $1 - e^{-\Omega(m)}$
2. The global optimal solution is not obtained with probability $1 - m^{-\Omega(m)}$

Then we state that only a direct jump from the local optimum to the global optimum is possible, and the probability of this event is $O(m^{-m/2})$.

First we show that with high probability $S(\pi_{init}) \leq \varepsilon m$ holds for the initial solution π_{init}, where ε is a small positive constant. We count the number of permutations in which at least εm, $\varepsilon > 0$ a small constant, of cluster-neighbourhoods are correct. We should select εm of the clusters to be followed by their specific neighbour, and consider the number of different permutations of $m - \varepsilon m$ clusters:

$$\binom{m}{\varepsilon m}(m - \varepsilon m)! \tag{4.1}$$

Some solutions are double-counted in this expression, so the actual number of different solutions with $S(\pi) \geq \varepsilon m$ is less than (4.1). Therefore, the probability of having more than εm clusters followed by their specific cluster, is at most

$$\binom{m}{\varepsilon m}\frac{(m - \varepsilon m)!}{m!} = ((\varepsilon m)!)^{-1} = O\left(\left(\frac{\varepsilon m}{2}\right)^{-\frac{\varepsilon m}{2}}\right)$$

Hence, with probability $1 - O((\frac{\varepsilon m}{2})^{-\frac{\varepsilon m}{2}})$, $S(\pi_{init}) \leq \varepsilon m$ holds and the initial solution has at at most εm correctly ordered clusters.

Now we analyse the expected time to reach a solution π with $S(\pi) = 0$. The probability of a good ordering to change to a bad one is at least

$$\left(\frac{1}{e}\right) \cdot \left(\frac{k}{m}\right) \cdot \left(\frac{m^2 - m}{m^2}\right)^k$$

where k is the number of edges which can be changed in each operation. For *jump* operation k equals 3. For all $m > 2$, it holds that $m < \frac{m^2}{2}$, so the probability above is at least

$$\left(\frac{1}{e}\right) \cdot \left(\frac{3}{m}\right) \cdot \left(\frac{1}{2}\right)^3 = \Omega(m^{-1})$$

Therefore, the expected time for each edge to be replaced with a bad edge is in $O(m)$ and for m edges it is in $O(m^2)$.

Now we consider a phase of $T = Cm^3$ iterations and show that the local optimum is reached with high probability. Let $C = 2C'$ and consider a phase of $2C'm^2$ iterations while assuming that the local optimum is expected to be reached in time $C'm^2$. Then by means of Markov's inequality we have

$$\Pr(T' > 2C'm^2) \leq \frac{1}{2}.$$

Repeating this m times, the probability of not reaching the local op-

timum is 2^{-m}. Therefore, the algorithm reaches the local optimum with probability $1 - 2^{-m} = 1 - e^{-\Omega(m)}$ during the phase of $T = Cm^3$ steps.

To prove that with high probability, the global optimum is not reached during the considered phase, note first that by Lemma 18, any jump to a solution closer to the optimum other than directly to the Global Optimum will be rejected.

Furthermore, for the initial solution $S(\pi_{init}) \leq \varepsilon m$. Therefore, only non-optimal solutions π with $S(\pi) \leq \varepsilon m$ are accepted by the algorithm. In order to obtain an optimal solution the algorithm has to produce the optimal solution from a solution π with $S(\pi) \leq \varepsilon m$ in a single mutation step. We now bound the probability of such a direct jump which changes at least $(1-\varepsilon)m$ clusters to their correct order from above. Such a move needs at least $\frac{(1-\varepsilon)m}{3}$ operations in the same iteration. Taking into account that these *Jump* operations may be acceptable in any order, the probability of a direct jump is at most

$$\frac{1}{e\left(\frac{(1-\varepsilon)m}{3}\right)!} \cdot \frac{1}{m^{\frac{(1-\varepsilon)m}{2}}} \cdot \left(\frac{(1-\varepsilon)m}{3}\right)! = m^{-\Omega(m)}. \tag{4.2}$$

So in a phase of $O(m^3)$ iterations the probability of having such a direct jump is by union bound at most $m^{-\Omega(m)+3} = m^{-\Omega(m)}$. So far we have shown that a local optimum π with $S(\pi) = 0$ is reached with probability $1 - e^{-\Omega(m)}$ within the first $T = Cm^3$ iterations. The probability of obtaining an optimal solution from a solution π with $S(\pi) = 0$ is at most

$$\frac{1}{e\left(\frac{m}{3}\right)!} \cdot \frac{1}{m^{\frac{m}{2}}} \cdot \left(\frac{m}{3}\right)! = e^{-1} \cdot m^{-\frac{m}{2}}$$

We now consider an additional phase of $(\frac{m}{2})^{\frac{m}{2}}$ steps after having obtained a local optimum. Using a union bound, the probability of reaching the global optimum in this phase is at most

$$\left(\frac{m}{2}\right)^{\frac{m}{2}} \cdot e^{-1} \cdot m^{-\frac{m}{2}} \leq \left(\frac{1}{2}\right)^{\frac{m}{2}}.$$

As a result, the probability of not reaching the optimal solution in these $(\frac{m}{2})^{\frac{m}{2}}$ iterations is $1 - 2^{-\frac{m}{2}} = 1 - e^{-\Omega(m)}$. Altogether, the runtime is at least $(\frac{m}{2})^{\frac{m}{2}}$ with probability $1 - e^{-\Omega(m)}$. □

4.4 Discussion of Generalisations

For a given solution visited in the upper level problem, the evaluation cost is, in the worst case, the computational complexity of the lower level problem. If the lower level problem can be solved in polynomial time, then a fixed-parameter bound on the the size of the upper level solution is sufficient to obtain a fixed-parameter tractable algorithm. For when the size of the upper level solution depends only on a parameter k of the original problem and not on its size n, any global random search heuristic on the upper level problem will be able to find the optimal upper level solution in no more than $f(k)$ iterations for some function $f(k)$ and will make $f(k) \cdot \mathrm{poly}(n)$ basic operations in total.

In our case, the *Global structure representation* of GMSTP and GTSP with m clusters, the size of an upper level solution is bounded from above by m^2 since it is enough to indicate whether any two clusters are connected or not to precisely define a solution. On the other hand the spanned-nodes representation of GMSTP needs a size of $m \log(n)$ to represent *which node* is selected in *each cluster*. If the solution size is restricted by a parameter m, uniform random search on the bit-string of length $O(f(m))$ will find the optimal solution in $2^{O(f(m))}$ iterations in expectation. With *Global structure representation*, if we pick our solutions uniformly at random the probability of picking a unique optimal solution is $(1/2)^{m^2}$ which will occur in $O(2^{m^2})$ time in expectation while uniform random search with the spanned node representation takes $\Omega(n^m)$ trials in expectation.

Summary

Evolutionary bi-level optimisation has gained an increasing interest in recent years. Its structure which allows separate approaches to the different levels of the problem allows us to test the flexibility of function evaluations for the evolutionary algorithms. This chapter contributes to the theoretical understanding by considering two classical NP-hard combinatorial optimisation problems, namely the generalized minimum spanning tree problem and the generalized travelling salesperson problem and showing how more involved fitness evaluation methods can reduce problems with bi-level structures to simpler search spaces.

Using parameterised computational complexity analysis of evolutionary algorithms for the generalized minimum spanning tree problem, we have examined two representations for the upper level solutions and used deterministic algorithms for lower level problems. Our results show that the *Global Structure Representation* leads to fixed parameter evolutionary algorithms. By presenting hard instances for each of the two approaches, we have pointed out where they run into difficulties. Furthermore, we have shown that the two representations for the generalized minimum spanning tree problem are complementary by proving that they are highly efficient on the hard instance of the other algorithm. After having achieved these results for the generalized minimum spanning tree problem, we turned our attention to the generalized travelling salesperson problem. We showed that the global structure representation leads to fixed parameter evolutionary algorithms with respect to the number of clusters. Furthermore, we pointed out a worst case instance where the runtime grows exponentially with respect to the number of clusters and discussed generalizations of the results to other bi-level optimisation problems.

Bi-level optimisation problems have a clearly separated structure which hints where a fitness evaluation method might fit. The effect of the fitness evaluation method on the performance of the resulting algorithm is easily analysed. However, when fitness evaluation methods are used for regular (not bi-level) problems, its effect on the runtime will be more complicated. In the following chapter we will provide a new analytical tool which allows a larger degree of freedom for the choice of fitness evaluation methods for regular problems.

Chapter 5

Runtime analysis of Population-Based Algorithms

5.1 Introduction

The theoretical understanding of Evolutionary Algorithms (EAs) has advanced significantly. A contributing factor for this success may have been the strategy to analyse simple settings before proceeding to more complex scenarios, while at the same time developing appropriate analytical techniques. In order to investigate further into the use of fitness evaluation methods, we will provide a new framework for analysing the progress of an evolutionary search process. This new framework aims at analysing the progress over different levels which divides the search space according to the proximity to a target subset (i.e. optimal solutions) rather than the progressive increase in the optimised fitness function. This level-based analysis will provide the tools to prove the runtime of algorithms which use alternative fitness evaluation methods that will guide the search process.

The fitness-level technique is one of the oldest techniques for deriving upper bounds on the expected runtime of EAs. In this technique, the solution space is partitioned into disjoint subsets called *fitness-levels* according to ascending values of the fitness function. The expected runtime can be deduced from bounds on the probabilities of escaping the fitness levels. Applications of the technique are widely known in the literature for classical *elitist* EAs [84]. In Section 3.2 we discussed how the fitness level technique evolved for population-based algorithms. Eremeev used a fitness-level technique to obtain bounds on the expected propor-

tion of the population of a *non-elitist* EA above a certain fitness level [33]. By generalising results in [63], the first adaptation of the fitness-level technique to run-time analysis of non-elitist population-based EAs was made by Lehre [60], and refined by Lehre and Dang [12]. One limitation of the previous approaches is that the partition must be fitness-based and only *unary variation operators* are allowed, e.g. Genetic Algorithms (GAs) are excluded.

Here, we show that the above limitations can be removed. This gives rise to a much more general tool which is applicable to a wide range of search processes involving non-elitist populations. As introductory examples, we analyse the runtime of variants of the Genetic Algorithm (GA) with different selection mechanisms and crossover operators on well-known functions, such as ONEMAX and LEADINGONES, and on the sorting problem.

The chapter is structured as follows. In Section 5.2 we present the general scheme of the algorithms covered by the main result of the chapter and show how the GAs fit as special cases into this scheme. Section 5.3 contains the main result of the chapter. Applications of the main result to different GAs are considered in Section 5.4. Concluding remarks are given in Section 5.5.

5.2 Algorithmic Scheme

We consider population-based algorithms at a very abstract level in which fitness evaluations, selection and variation operations, which depending on the current population P of size λ, are represented by a distribution $D(P)$ over a finite set $\mathcal{X}$. More precisely, D is a mapping from $\mathcal{X}^\lambda$ into the space of probability distributions over $\mathcal{X}$. The next generation is obtained by sampling each new individual independently from $D(P)$. This scheme is summarised below.

Algorithm 7 Population-based algorithm with independent sampling

Require:
Finite state space $\mathcal{X}$, and population size $\lambda \in \mathbb{N}$,
Mapping D from $\mathcal{X}^\lambda$ to the space of probability distributions over $\mathcal{X}$.

1: $P_0 \sim \mathrm{Unif}(\mathcal{X}^\lambda)$
2: **for** $t = 0, 1, 2, \ldots$ until termination condition met **do**
3: Sample $P_{t+1}(i) \sim D(P_t)$ independently for each $i \in [\lambda]$
4: **end for**

A similar scheme was studied in [83], where it was called *Random Heuristic Search* with *an admissible transition rule* (see [83]). Some examples of such algorithms are Simulated Annealing (more generally any algorithm with the population composed of a single individual), Stochastic Beam Search [83], Estimation of Distribution Algorithms such as the Univariate Marginal Distribution Algorithm [5] and the Genetic Algorithm (GA) [39]. The previous studies of the framework were often limited to some restricted settings [72] or mainly focused on infinite populations [83]. Here, we are interested in finite populations and develop a general method to deduce the expected runtime of the search processes defined in terms of *number of evaluations*. We illustrate our methods with runtime analysis of GAs under various settings (which are different to [72]).

The term Genetic Algorithm (GA) is often applied to EAs that use recombination operators. The GA is Algorithm 7 where the sampling $y \sim D(P_t)$ is the following: (i) $u \sim p_{\text{sel}}(P_t)$, $v \sim p_{\text{sel}}(P_t)$ (selection); (ii) $\{x', x''\} \sim p_{\text{xor}}(u, v)$, $x \sim \text{Unif}(\{x', x''\})$ (crossover); (iii) $y \sim p_{\text{mut}}(x)$ (mutation). The additional part $x \sim \text{Unif}(\{x', x''\})$ at crossover is to match Algorithm 7 that produces only one resulting bit-string. We call this operator the *one-offspring* version of the *standard* crossover. In the rest of this chapter, the two operations of the one-offspring version will be denoted simply by $x \sim p_{\text{xor}}(x, y)$. Here the standard operators of GA are formally represented by transition matrices:

- $p_{\text{sel}} : [\lambda] \to [0, 1]$ represents a selection operator, where $p_{\text{sel}}(i|P_t)$ is the probability of selecting the i-th individual from population P_t.
- $p_{\text{mut}} : \mathcal{X} \times \mathcal{X} \to [0, 1]$, where $p_{\text{mut}}(y|x)$ is the probability of mutating $x \in \mathcal{X}$ into $y \in \mathcal{X}$.
- $p_{\text{xor}} : \mathcal{X} \times \mathcal{X}^2 \to [0, 1]$, where $p_{\text{xor}}(x|u, v)$ is the probability of obtaining x as a result of crossover (or recombination) between $u, v \in \mathcal{X}$

5.3 Main Theorem

This section states a general technique for obtaining upper bounds on the expected runtime of any process that can be described in the form of Algorithm 7.

Suppose that for some m there is an ordered partition of $\mathcal{X}$ into subsets $(A_1, \ldots, A_{m+1})$ called *levels*. For $j \in [m]$ we denote by $A_j^+ := \cup_{i=j+1}^{m+1} A_i$, the union of all levels above level j. An example of partition is the *canonical* partition, where each level regroups solutions having the same fitness value (see e.g. [60]). This partition is classified as *fitness-based*, our main theorem is not limited to this particular type of partition.

Theorem 20. *Given a partition $(A_1, \ldots, A_{m+1})$ of $\mathcal{X}$, define $T := \min\{t\lambda \mid |P_t \cap A_{m+1}| > 0\}$ to be the first point in time that elements of A_{m+1} appear in P_t of Algorithm 7. If there exist parameters $z_1, \ldots, z_m, z_* \in (0, 1]$, $\delta > 0$, a constant $\gamma_0 \in (0, 1)$ and a function $z_0 : (0, \gamma_0) \to \mathbb{R}$ such that for all $j \in [m]$, $P \in \mathcal{X}^\lambda$, $y \sim D(P)$ and $\gamma \in (0, \gamma_0)$ we have*

(G1) $\Pr\left(y \in A_j^+ \mid |P \cap A_{j-1}^+| \geq \gamma_0\lambda\right) \geq z_j \geq z_*$

(G2) $\Pr\left(y \in A_j^+ \mid |P \cap A_{j-1}^+| \geq \gamma_0\lambda, |P \cap A_j^+| \geq \gamma\lambda\right) \geq z_0(\gamma) \geq (1+\delta)\gamma$

(G3) $\lambda \geq \frac{2}{a} \ln\left(\frac{16m}{ac\varepsilon z_*}\right)$ *with* $a = \frac{\delta^2\gamma_0}{2(1+\delta)}$, $\varepsilon = \min\{\delta/2, 1/2\}$ *and* $c = \varepsilon^4/24$

then $E[T] \leq \frac{2}{c\varepsilon}\left(m\lambda(1 + \ln(1 + c\lambda)) + \sum_{j=1}^{m} \frac{1}{z_j}\right)$

Informally, the two first conditions require a relationship between P and the distribution $D(P)$: (G1) demands a certain probability z_j of creating an individual at level $j+1$ or better when some fixed portion of the population is already at level j (or higher); (G2) requires that in the fixed portion, the number of individuals at levels strictly higher than j (if those exist) tends to increase, e.g. by a multiplicative factor of $1 + \delta$. Finally, (G3) requires a sufficiently large population size. The proof follows the same ideas as those in [12].

Proof of Theorem 20. We use the following notation. The number of individuals in $A_j \cup A_j^+$ at generation t is denoted by X_t^j. The current level of the population at generation t is denoted by Z_t, where $Z_t := \ell$ iff $X_t^\ell \geq \lceil \gamma_0\lambda \rceil$ and $X_t^{\ell+1} < \gamma_0\lambda$. Note that Z_t is uniquely defined, as it is the level of the γ_0-ranked individual at generation t. We also use q_j to denote the probability to generate at least one individual at a level strictly greater than j in the next generation, knowing that there are at least $\lceil \gamma_0\lambda \rceil$ individuals of the population at level j or higher in the current generation. Because of (G1), we have $q_j \geq 1 - (1 - z_j)^\lambda \geq z_j\lambda/(z_j\lambda + 1)$.

The theorem can now be proved using the additive drift theorem [43] with respect to the potential function $g(t) := g_1(t) + g_2(t)$, where

$$g_1(t) := (m - Z_t)\ln(1 + c\lambda) - \ln(1 + cX_t^{Z_t+1})$$
$$\text{and } g_2(t) := \frac{1}{q_{Z_t}e^{\kappa X_t^{Z_t+1}}} + \sum_{j=Z_t+1}^{m-1}\frac{1}{q_j} \text{ with } \kappa \in (0, \delta)$$

The above components originated from the drift analysis of [60], then later improved by [12]. Function g is bounded from above by, $g(t) \leq m\ln(1 + c\lambda) + \sum_{j=1}^{m}\frac{1}{q_j} \leq m(1 + \ln(1 + c\lambda)) + \frac{1}{\lambda}\sum_{j=1}^{m}\frac{1}{z_j}$. At generation t, we use $R = Z_{t+1} - Z_t$ to denote the random variable describing the next progress in terms of levels. To simplify further writing, let us put $\ell = Z_t$, $i = X_t^\ell$, $X = X_{t+1}^\ell$, then $\Delta = g(t) - g(t+1) = \Delta_1 + \Delta_2$ with

$$\Delta_1 := g_1(t) - g_1(t+1) = R\ln(1 + c\lambda) + \ln\left(\frac{1 + X_{t+1}^{\ell+R+1}}{1 + ci}\right)$$
$$\Delta_2 := g_2(t) - g_2(t+1) = \frac{1}{q_\ell e^{\kappa i}} - \frac{1}{q_{\ell+R}e^{\kappa X_{t+1}^{\ell+R+1}}} + \sum_{j=\ell+1}^{l+R}\frac{1}{q_j}$$

Let us denote by $\mathcal{E}_t$ the event that the population in the next generation does not fall down to a lower level, $\mathcal{E}_t : Z_{t+1} \geq Z_t$. We first compute the *conditional forward drift* $E[\Delta|\mathscr{F}_t, \mathcal{E}_t]$, here $\mathscr{F}_t$ is the filtration induced by P_t. Under $\mathcal{E}_t$, R is a non-negative random variable and Δ is a random variable indexed by R, noted as $\Delta = Y_R$. We can show that $Y_{r\geq 1} \geq Y_0$ for fixed indexes.

$$Y_0 = \ln\left(\frac{1 + cX}{1 + ci}\right) + \frac{1}{q_\ell e^{\kappa i}} - \frac{1}{q_\ell e^{\kappa X}} \leq \ln\left(\frac{1 + c\lambda}{1 + ci}\right) + \frac{1}{q_\ell e^{\kappa i}}$$
$$Y_{r\geq 1} = r\ln(1 + c\lambda) + \ln\left(\frac{1 + X_{t+1}^{\ell+r+1}}{1 + ci}\right) + \frac{1}{q_\ell e^{\kappa i}} - \frac{1}{q_{\ell+r}e^{\kappa X_{t+1}^{\ell+r+1}}} + \sum_{j=\ell+1}^{\ell+r}\frac{1}{q_j}$$
$$\geq \ln(1 + c\lambda) + \ln\left(\frac{1}{1 + ci}\right) + \frac{1}{q_\ell e^{\kappa i}} - \frac{1}{q_{\ell+r}} + \sum_{j=\ell+1}^{\ell+r}\frac{1}{q_j}$$
$$= \ln\left(\frac{1 + c\lambda}{1 + ci}\right) + \frac{1}{q_\ell e^{\kappa i}} + \sum_{j=\ell+1}^{\ell+r-1}\frac{1}{q_j} \geq \ln\left(\frac{1 + c\lambda}{1 + ci}\right) + \frac{1}{q_\ell e^{\kappa i}} \geq Y_0$$

It follows from Lemma 34 that $E[\Delta|\mathscr{F}_t, \mathcal{E}_t] = E[Y_R|\mathscr{F}_t, \mathcal{E}_t] \geq E[Y_0|\mathscr{F}_t, \mathcal{E}_t]$, so we only focus on $r = 0$ to bound the drift from below. We separate two cases, $i = 0$, the event is denoted by $\mathcal{Z}_t$, and $i \geq 1$ (event $\bar{\mathcal{Z}}_t$). Re-

call that each individual is generated independently from each other, so during $\bar{\mathcal{Z}}_t$ we have that $X \sim \mathrm{Bin}(\lambda, p)$ where $p \geq z_0(i/\lambda)$. From (G2), we also get $z_0(i/\lambda) \geq (i/\lambda)(1+\delta)$. Hence $p \geq (i/\lambda)(1+\delta)$ and by Lemmas 32, 33, it holds for $i \geq 1$ (event $\bar{\mathcal{Z}}_t$) that

$$E\left[\Delta_1|\mathscr{F}_t, \mathcal{E}_t, \bar{\mathcal{Z}}_t\right] \geq E\left[\ln\left(\frac{1+cX}{1+ci}\right)|\mathscr{F}_t, \mathcal{E}_t, \bar{\mathcal{Z}}_t\right] \geq c\varepsilon$$
$$E\left[\Delta_2|\mathscr{F}_t, \mathcal{E}_t, \bar{\mathcal{Z}}_t\right] \geq \frac{1}{q_\ell}(e^{-\kappa i} - E\left[e^{-\kappa X}|\mathscr{F}_t, \mathcal{E}_t, \bar{\mathcal{Z}}_t\right]) \geq 0$$

For $i = 0$, we get $E\left[\Delta_1|\mathscr{F}_t, \mathcal{E}_t, \mathcal{Z}_t\right] \geq E\left[\ln(1)|\mathscr{F}_t, \mathcal{E}_t, \mathcal{Z}_t\right] = 0$ because $X \geq 0$. Recall that $q_\ell = \Pr\left(X \geq 1|\mathscr{F}_t, \mathcal{E}_t, \mathcal{Z}_t\right)$, so

$$E\left[\Delta_2|\mathscr{F}_t, \mathcal{E}_t, \mathcal{Z}_t\right] \geq \Pr\left(X \geq 1|\mathscr{F}_t, \mathcal{E}_t, \mathcal{Z}_t\right) E\left[\frac{1}{q_\ell e^{\kappa i}} - \frac{1}{q_\ell e^{\kappa X}}|\mathscr{F}_t, \mathcal{E}_t, \mathcal{Z}_t, X \geq 1\right]$$
$$\geq q_\ell(1/q_\ell)(e^{-\kappa\cdot 0} - e^{-\kappa\cdot 1}) = 1 - e^{-\kappa}$$

So the conditional forward drift is $E\left[\Delta|\mathscr{F}_t, \mathcal{E}_t\right] \geq \min\{c\varepsilon, 1-e^{-\kappa}\}$. Furthermore, κ can be picked in the non-empty interval $(-\ln(1-c\varepsilon), \delta) \subset (0, \delta)$, so that $1 - e^{-\kappa} > c\varepsilon$ and $E\left[\Delta|\mathscr{F}_t, \mathcal{E}_t\right] \geq c\varepsilon$. Next, we compute the *conditional backward drift*, which can be done for the worst case.

$$E\left[\Delta|\mathscr{F}_t, \bar{\mathcal{E}}_t\right] \geq -(m-1)\ln(1+c\lambda) - \ln(1+c\lambda) - \sum_{j=1}^{m} 1/q_j \geq -m\left(c\lambda + 2/z_*\right)$$

The probability that event $\mathcal{E}_t$ does not occur is computed as follows. Recall that $X_t^\ell \geq \lceil\gamma_0\lambda\rceil$ and X_{t+1}^ℓ is binomially distributed random variable with probability at least $z_0(\gamma_0) \geq (1+\delta)\gamma_0$ by condition (G2), so $E\left[X_{t+1}^\ell|\mathscr{F}_t\right] \geq (1+\delta)\gamma_0\lambda$. The event $\bar{\mathcal{E}}_t$ happens when the number of individuals at level ℓ is strictly less than $\lceil\gamma_0\lambda\rceil$ in the next generation. By a Chernoff bound (Theorem 8), we have

$$\begin{aligned}
\Pr\left(\bar{\mathcal{E}}_t|\mathscr{F}_t\right) &= \Pr\left(X_{t+1}^\ell < \lceil\gamma_0\lambda\rceil|\mathscr{F}_t\right) \leq \Pr\left(X_{t+1}^\ell \leq \gamma_0\lambda|\mathscr{F}_t\right) \\
&= \Pr\left(X_{t+1}^\ell \leq (1-\delta/(1+\delta))\,(1+\delta)\gamma_0\lambda|\mathscr{F}_t\right) \\
&\leq \Pr\left(X_{t+1}^\ell \leq (1-\delta/(1+\delta))\,E\left[X_{t+1}^\ell|\mathscr{F}_t\right]|\mathscr{F}_t\right) \\
&\leq \exp\left(-\frac{\delta^2 E\left[X_{t+1}^\ell|\mathscr{F}_t\right]}{2(1+\delta)^2}\right) \leq \exp\left(-\frac{\delta^2(1+\delta)\gamma_0\lambda}{2(1+\delta)^2}\right) = e^{-a\lambda}
\end{aligned}$$

Recall condition (G3) that $\lambda \geq (2/a)\ln((16m)/(ac\varepsilon z_*))$. This implies that $(8m)/(ac\varepsilon z_*) \leq e^{\frac{a\lambda}{2}}/2 \leq e^{a\lambda}/(a\lambda)$ (see Lemma 31), or $e^{-a\lambda} \leq c\varepsilon z_*/(8m\lambda)$.

So,

$$\begin{aligned}
E\left[\Delta|\mathscr{F}_t\right] &= (1-\Pr\left(\bar{\mathcal{E}}_t|\mathscr{F}_t\right))E\left[\Delta|\mathscr{F}_t,\mathcal{E}_t\right] + \Pr\left(\bar{\mathcal{E}}_t|\mathscr{F}_t\right)E\left[\Delta|\mathscr{F}_t,\bar{\mathcal{E}}_t\right] \\
&= E\left[\Delta|\mathscr{F}_t,\mathcal{E}_t\right] - \Pr\left(\bar{\mathcal{E}}_t|\mathscr{F}_t\right)\left(E\left[\Delta|\mathscr{F}_t,\mathcal{E}_t\right] - E\left[\Delta|\mathscr{F}_t,\bar{\mathcal{E}}_t\right]\right) \\
&\geq c\varepsilon - \frac{c\varepsilon z_*}{8m\lambda}\left(c\varepsilon + m\left(c\lambda + \frac{2}{z_*}\right)\right) \\
&\geq c\varepsilon - \frac{c\varepsilon}{8}\left(\frac{c\varepsilon z_*}{\lambda m} + z_* c + \frac{2}{\lambda}\right) \geq c\varepsilon - \frac{4c\varepsilon}{8} = \frac{c\varepsilon}{2}
\end{aligned}$$

Theorem 6 now gives $E\left[T\right] \leq \frac{2}{c\varepsilon}\left(m\lambda(1+\ln(1+c\lambda)) + \sum_{j=1}^{m}\frac{1}{z_j}\right)$.

□

In the special case of unary variation operators Theorem 20 becomes analogous to the main results of [12, 60]. It is an open problem whether the upper bound in Theorem 20 is tight. The lack of general tools make this problem hard. The family tree technique [86], the population drift theorem [59], and the fitness level technique in [78], provide lower bounds for population-based algorithms, however only for less general settings than Algorithm 7.

5.4 Runtime Analysis of Genetic Algorithms

This section provides a version of Theorem 20 tailored to the GAs described in Section 5.2. The *selective pressure* of a selection mechanism p_{sel} is defined as follows. For any $\gamma \in (0,1)$ and population P of size λ, let $\beta(\gamma, P)$ be the probability of selecting an individual from P that is at least as good as the individual with rank $\lceil\gamma\lambda\rceil$ (see [12] or [60] for a formal definition). We assume that p_{sel} is *monotone* with respect to fitness values [60], ie for all $P \in \mathcal{X}^\lambda$ and pairs $i, j \in [\lambda]$, $p_{\text{sel}}(i \mid P) \geq p_{\text{sel}}(j \mid P)$ if and only if $f(P(i)) \geq f(P(j))$.

Corollary 21. *Given a function $f : \mathcal{X} \to \mathbb{R}$ and a partition $(A_1, \ldots, A_{m+1})$ of $\mathcal{X}$, let $T := \min\{t\lambda \mid |P_t \cap A_{m+1}| > 0\}$ be the runtime of the non-elitist Genetic Algorithm, as described in Section 5.2, on f. If there exist parameters $s_1, \ldots, s_m, s_*, p_0, \varepsilon_1 \in (0,1]$, $\delta > 0$, and a constant $\gamma_0 \in (0,1)$ such that for all $j \in [m]$, $P \in \mathcal{X}^\lambda$, and $\gamma \in (0, \gamma_0)$*

(C1) $p_{\text{mut}}(y \in A_j^+ \mid x \in A_{j-1}^+) \geq s_j \geq s_*$
(C2) $p_{\text{mut}}(y \in A_j^+ \mid x \in A_j^+) \geq p_0$

(C3) $p_{\text{xor}}(x \in A_j^+ \mid u \in A_{j-1}^+, v \in A_j^+) \geq \varepsilon_1$

(C4) $\beta(\gamma, P) \geq \gamma\sqrt{\frac{1+\delta}{p_0\varepsilon_1\gamma_0}}$

(C5) $\lambda \geq \frac{2}{a}\ln\left(\frac{32mp_0}{(\delta\gamma_0)^2 cs_*\psi}\right)$ *with* $a := \frac{\delta^2\gamma_0}{2(1+\delta)}$, $\psi := \min\{\frac{\delta}{2}, \frac{1}{2}\}$ *and* $c := \frac{\psi^4}{24}$

then $E[T] \leq \frac{2}{c\psi}\left(m\lambda(1+\ln(1+c\lambda)) + \frac{p_0}{(1+\delta)\gamma_0}\sum_{j=1}^{m}\frac{1}{s_j}\right)$.

Proof. We show that conditions (C1-5) imply conditions (G1-3) in Theorem 20.We first show that condition (G1) is satisfied for $z_j = \gamma_0(1+\delta)s_j/p_0$. Assume that $|P \cap A_{j-1}^+| \geq \gamma_0\lambda$. Note that (C3) written for one level below, which is $p_{\text{xor}}(x \in A_{j-1}^+ \mid u \in A_{j-2}^+, v \in A_{j-1}^+) \geq \varepsilon_1$, implies $p_{\text{xor}}(x \in A_{j-1}^+ \mid u \in A_{j-1}^+, v \in A_{j-1}^+) \geq \varepsilon_1$. To sample an individual in A_j^+, it suffices that the selection operator picks two individuals u and v from A_{j-1}^+, that the crossover operator produces an individual x in A_{j-1}^+ from u and v, and the mutation operator produces an individual y in A_j^+ from x. By conditions (C4), (C3) as the remark, and (C1), the probability of this event is at least $\beta(\gamma_0)\beta(\gamma_0)\varepsilon_1 s_j \geq \gamma_0(1+\delta)s_j/p_0 = z_j$.

We then show that condition (G2) is satisfied. Assume that $|P \cap A_{j-1}^+| \geq \gamma_0\lambda$ and $|P \cap A_j^+| \geq \gamma\lambda$. To produce an individual y in A_j^+, it suffices that the selection operator picks an individual u in A_{j-1}^+ and an individual v in A_j^+, that the crossover operator produces an individual x in A_j^+ from u and v, and the mutation operator produces an individual y in A_j^+ from x. By conditions (C4), (C3), and (C2), the probability of this event is at least $\beta(\gamma_0)\beta(\gamma)\varepsilon_1 p_0 \geq (1+\delta)\gamma$.

Finally, to see that condition (G3) is satisfied, it suffices to note that $az_* = (\delta\gamma_0)^2 s_*/(2p_0)$. Hence, the statement now follows from Theorem 20. □

5.4.1 Runtime of GAs on Simple pseudo-Boolean Functions

We apply Corollary 21 to bound the expected runtime of the non-elitist GA on the functions $\textsc{OneMax}(x) := \sum_{i=1}^{n} x_i$ (also written shortly as $|x|_1$ or OM) and $\textsc{LeadingOnes}(x) := \sum_{i=1}^{n}\prod_{j=1}^{i} x_j$ (shortly as LO).

We first show how to parameterise three standard selection mechanisms such that condition (C4) is satisfied. In *k-tournament selection*, k individuals are sampled uniformly at random with replacement from

the population, and the fittest of these individuals is returned. In (μ, λ)-*selection*, parents are sampled uniformly at random among the fittest μ individuals in the population. A function $\alpha : \mathbb{R} \to \mathbb{R}$ is a ranking function [39] if $\alpha(x) \geq 0$ for all $x \in [0, 1]$, and $\int_0^1 \alpha(x)dx = 1$. In ranking selection with ranking function α, the probability of selecting individuals ranked γ or better is $\int_0^\gamma \alpha(x)dx$. We define *exponential ranking* parameterised by $\eta > 0$ as $\alpha(\gamma) := \eta e^{\eta(1-\gamma)}/(e^\eta - 1)$.

Lemma 22. *For any constant $\delta > 0$, there exists a constant $\gamma_0 \in (0, 1)$ such that*

1. *k-tournament selection with $k \geq 4(1 + \delta)/(\varepsilon_1 p_0)$ satisfies (C4)*
2. *(μ, λ)-selection with $\lambda/\mu \geq (1 + \delta)/(\varepsilon_1 p_0)$ satisfies (C4)*
3. *exponential ranking selection with $\eta \geq 4(1 + \delta)/(\varepsilon_1 p_0)$, satisfies (C4).*

Proof. Let $\varepsilon := \varepsilon_1 p_0$. For k-tournament selection with $k \geq 4(1 + \delta)/\varepsilon$, we have

$$\beta(\gamma) \geq 1 - \frac{1}{\gamma k + 1} \geq 1 - \frac{1}{\gamma 4(1 + \delta)/\varepsilon + 1} = \frac{\gamma 4(1 + \delta)/\varepsilon}{\gamma 4(1 + \delta)/\varepsilon + 1}$$

If $\gamma_0 := \varepsilon/(4(1 + \delta))$, then for all $\gamma \in (0, \gamma_0]$ it holds that $4(1 + \delta)/\varepsilon \leq 1/\gamma$ and

$$\beta(\gamma) \geq \frac{\gamma 4(1 + \delta)/\varepsilon}{\gamma(1/\gamma) + 1} = \frac{2(1 + \delta)\gamma}{\varepsilon} = \sqrt{\frac{(1 + \delta)}{\varepsilon(\varepsilon/4(1 + \delta))}}\gamma = \sqrt{\frac{(1 + \delta)}{\varepsilon\gamma_0}}\gamma$$

$\beta(\gamma_0)\beta(\gamma) \geq (1 + \delta)\gamma/\varepsilon$ for all $\gamma \in (0, \gamma_0]$. In (μ, λ)-selection, for all $\gamma \in (0, \mu/\lambda]$ we have $\beta(\gamma) = \lambda\gamma/\mu$. It suffices to pick $\gamma_0 := \mu/\lambda$ so that with $\lambda/\mu \geq (1 + \delta)/\varepsilon$, for all $\gamma \in (0, \gamma_0]$

$$\beta(\gamma) \geq \frac{\lambda\gamma}{\mu} = \sqrt{\frac{\lambda^2}{\mu^2}}\gamma = \sqrt{\frac{\lambda}{\mu\gamma_0}}\gamma \geq \sqrt{\frac{1 + \delta}{\varepsilon\gamma_0}}\gamma$$

In exponential ranking selection, we have

$$\beta(\gamma) \geq \int_0^\gamma \frac{\eta e^{\eta(1-x)}dx}{e^\eta - 1} = \left(\frac{e^\eta}{e^\eta - 1}\right)\left(1 - \frac{1}{e^{\eta\gamma}}\right) \geq 1 - \frac{1}{1 + \eta\gamma}$$

The rest of the proof is similar to tournament selection with η in place of k, e.g. based on the input condition on η, it suffices to pick $\gamma_0 := \varepsilon_1 p_0/(4(1 + \delta))$.

□

Lemma 23 shows that two standard crossover operators satisfy (C3) for $\varepsilon_1 = \frac{1}{2}$.

Lemma 23. *If $x \sim p_{\text{xor}}(u, v)$, where p_{xor} is one-point or uniform crossover, then*

1. *If $\text{LO}(u) = \text{LO}(v) = j$, then $\Pr(\text{LO}(x) \geq j) = 1$.*
2. *If $\text{LO}(u) \neq \text{LO}(v)$, then $\Pr(\text{LO}(x) > \min\{\text{LO}(u), \text{LO}(v)\}) \geq 1/2$.*
3. *$\Pr(\text{OM}(x) \geq \lceil(\text{OM}(u) + \text{OM}(v))/2\rceil) \geq 1/2$.*

Proof. (i) For the standard version of both crossover operators with two offspring, at any bit-position $i \in [n]$, the bit-value of the output x_i is equal to either u_i or w_i. Hence, the two resulting bit-strings have j leading ones and so does the result of the one-offspring crossover with probability 1.

(ii) W.l.o.g. we can assume $\text{LO}(v) = j$ and $\text{LO}(u) > \text{LO}(v)$. Then v has a 0 while u has a 1 at position $j + 1$. It is clear that one of the resulting bit-strings in the standard version of crossover will inherit the 1 at that position and the other will inherit the 0. This implies that one of those resulting bit-strings has fitness at least $j + 1$ and with probability $1/2$ it is being selected in the one-offspring crossover.

(iii) Recall that x', x'' are the two bit-strings resulting from the standard one-point or uniform crossover operator with two offspring. Each bit of u and v is copied either to x' or to x''. So $|x'|_1 + |x''|_1 = |u|_1 + |v|_1$, which means that $\max\{|x'|_1, |x''|_1\} \geq \lceil(|u|_1 + |v|_1)/2\rceil$.

In the single-offspring crossover, x is chosen with probabilities $1/2$ to be copied either from x' or x''. Therefore, with probability at least $1/2$ we have $\text{OM}(x) \geq \lceil(|u|_1 + |v|_1)/2\rceil$. □

Theorem 24. *Assume that the non-elitist GA uses one-point or uniform crossover, bitwise mutation with mutation rate χ/n for a constant $\chi > 0$, and either k-tournament selection with $k \geq 8(1 + \delta)e^\chi$, or (μ, λ) selection with $\lambda/\mu \geq 2(1 + \delta)e^\chi$ or the exponential ranking selection with $\eta \geq 8(1 + \delta)e^\chi$, for a constant $\delta > 0$. Then there exists a constant $c > 0$, such that the GA with population size $\lambda \geq c \ln n$, has expected runtime $O(n\lambda \ln \lambda + n^2)$ on* LeadingOnes, *and expected runtime $O(n\lambda \ln \lambda)$ on* OneMax.

Proof. Let f be either OM or LO. We apply Corollary 21 with the canonical partition of the search space into $n+1$ levels $A_j := \{x \mid f(x) = j - 1\}$,

for $j \in [n+1]$. We use $p_0 := (1-\chi/n)^n$ the probability of not flipping any bit position by mutation, and for all $j \in [n]$, define

$$s_j := \begin{cases} (\chi/n)(1-\chi/n)^{n-1} & \text{if } f = \textsc{Lo}, \text{ and} \\ (n-j+1)(\chi/n)(1-\chi/n)^{n-1}p_0 & \text{if } f = \textsc{Om}. \end{cases}$$

Considering condition (C1), when $x \in A_j$ it suffices to upgrade x to a higher level, the probability of such an event is at least s_j for LO and $s_j/p_0 > s_j$ for OM. When $x \in A_j^+$, it suffices to not modify x, the probability of such an event is at least $p_0 \geq s_j$ with sufficiently large n for LO and $p_0 > (n-j+1)(\chi/n)(1-\chi/n)^{n-1}p_0 = s_j$ for OM. So, condition (C1) is satisfied for both functions with the given s_j. In addition, condition (C2) is trivially satisfied for the given p_0 and condition (C3) is satisfied for the parameter $\varepsilon_1 := 1/2$ by Lemma 23.

We now look at condition (C4), and remark that $p_0 = (1-\chi/n)^{(n/\chi-1)\chi}(1-\chi/n)^\chi \geq e^{-\chi}(1-\chi/n)^\chi$. So $e^\chi \geq (1-\chi/n)^\chi/p_0$ and with the given condition for k-tournament, we get $k \geq 8(1+\delta)e^\chi = 4(1+\delta)e^\chi/(1/2) \geq 4(1+\delta)(1-\chi/n)^\chi/(\varepsilon_1 p_0)$. Then for any constant $\delta' \in (0,\delta)$ and sufficiently large n, literally $n \geq \chi/(1-((1+\delta')/(1+\delta))^{1/\chi})$, it holds that $k \geq 4(1+\delta')/(\varepsilon_1 p_0)$. So condition (C4) is satisfied with the constant δ' for k-tournament by Lemma 22. The same reasoning can be applied so that (C4) is also satisfied for the other selection mechanisms.

Finally, $s_* := \min_{j\in[n]} s_j = \Omega(1/n)$. So assuming $\lambda \geq c \ln n$ for a sufficiently large constant c, condition (C5) is satisfied as well. Note that the p_0 part in s_j of LO only removes the p_0 from $p_0/(1+\delta)\gamma_0$ in the runtime of Corollary 21. Therefore, the upper bounds $O(n\lambda \ln \lambda + n^2)$ and $O(n\lambda \ln \lambda)$ on the expected runtime are proven for OM and LO respectively.

□

Note that the upper bounds in Theorem 24 match the upper bounds obtained in [12] for EAs without crossover.

5.4.2 Runtime of GAs on the Sorting Problem

Given n distinct elements from a totally ordered set, we consider the problem of finding an ordering of them so that some *measure of sortedness* is maximised. Scharnow et al. [77] considered several sortedness measures in the context of analysing the (1+1) EA. One of those is $INV(\pi)$

which is defined to be the number of pairs (i, j) such that $1 \leq i < j \leq n$, $\pi(i) < \pi(j)$ (i.e. pairs in correct order). We show that with the methods introduced in this paper, analysing GAs on $INV(\pi)$ is not much harder than analysing the (1+1) EA.

As mutation operator, we consider the $Exchange(\pi)$ operator, which consecutively applies N pairwise exchanges between uniformly selected pairs of indices, where N is a random number drawn from a Poisson distribution with parameter 1. We consider a crossover operator, denoted by $p_{\mathrm{xor}(p_c)}$, which returns one of the parents unchanged with probability $1-p_c$. For example, $p_{\mathrm{xor}(p_c)}$ is built up from any standard crossover operator so that with probability p_c the standard operator is applied and the offspring is returned, otherwise with probability $1-p_c$ one of the parents is returned in place of the offspring. This construction corresponds to a typical setting (see [39]) where there is some *crossover probability* p_c of applying the crossover before the mutation. As selection mechanism, we consider k-tournament selection, (μ, λ)-selection, and exponential ranking selection.

Theorem 25. *If the GA uses a $p_{\mathrm{xor}(p_c)}$ crossover operator with p_c being any constant in $[0, 1)$, the Exchange mutation operator where the number of exchanges N is drawn from a Poisson distribution with parameter 1, k-tournament selection with $k \geq 8e(1+\delta)/(1-p_c)$, or (μ, λ)-selection with $\lambda/\mu \geq 2e(1+\delta)/(1-p_c)$, or exponential ranking selection with $\eta \geq 8e(1+\delta)/(1-p_c)$, then there exists a constant $c > 0$ such that if the population size is $\lambda \geq c \ln n$, the expected time to obtain the optimum of INV is $O(n^2 \lambda \log \lambda)$.*

Proof. Define $m := \binom{n}{2}$. We apply Corollary 21 with the canonical partition, $A_j := \{\pi \mid INV(\pi) = j\}$ for $j \in [m]$. The probability that the exchange operator exchanges 0 pairs is $1/e$. Hence, condition (C2) is trivially satisfied for $p_0 := 1/e$.

To show that condition (C1) is satisfied, define first $s_j := (m-j)p_0/(em)$. In the case that $x \in A_j$, then the probability that the exchange operator exchanges exactly one pair is $1/e$, and the probability that this pair is incorrectly ordered in x, is $(m-j)/m$. In the other case that, $x \in A_j^+$, it is sufficient that the exchange operator exchanges 0 pairs, which by condition (C2) occurs with probability at least $p_0 \geq s_j$. Hence, in both cases, $y \in A_j^+$ with probability at least s_j, and condition (C1) is satisfied.

Condition (C3) is trivially satisfied for $\varepsilon_1 := (1-p_c)/2$, because the crossover operator returns one of the parents unchanged with probabil-

ity $1 - p_c$, and with probability $1/2$, this parent is v. Condition (C4) is satisfied for some constant $\gamma_0 \in (0, 1)$ by Lemma 22. Finally, since γ_0, δ, and p_0 are constants, there exists a constant $c > 0$ such that condition (C5) is satisfied for any $\lambda \geq c \ln(n)$.

It therefore follows that the expected runtime of the GA on INV is bounded from above by $O(n^2 \lambda \log \lambda)$.

□

5.5 Summary

Most results in runtime analysis of evolutionary algorithms concern relatively simple algorithms, e.g. the (1+1) EA, which do not employ populations and higher-arity variation operators, such as crossover. This paper introduces a new tool, akin to the fitness-level technique, that easily yields upper bounds on the expected runtime of complex, non-elitist search processes. A strength of this new tool is that it does not require fitness dependent search space partitioning. The tool is illustrated on Genetic Algorithms. Given an appropriate balance between selection and variation operators, we have shown that GAs optimise standard benchmark functions, as well as combinatorial optimisation problems, efficiently. The level-based theory presented in this chapter has been used in subsequent collaborations of Lehre with other authors to analyse various algorithms (such as EDAs [14] and self-adapting EAs [15]) and various settings (dynamic optimisation [11] and noisy optimisation [13]). In the following chapter, we will illustrate how an evolutionary algorithm with a tailored fitness evaluation method can be designed using the guidelines provided by the level-based theory.

Chapter 6

Theory-driven algorithmic design for shortest path problems

6.1 Introduction

This chapter presents a principled way of designing a genetic algorithm which guarantees a rigorously proven upper bound on its optimisation time. The shortest path problem is selected to demonstrate how a general purpose theoretical analysis tool (i.e level-based analysis presented in Chapter 5), can be used as a design guide. We exploit the freedom of choosing a fitness independent level-structure and tailor a solution comparison method which replaces fitness evaluations. Apart from providing an example of theory-driven algorithmic design, we also provide the first runtime analysis of a non-elitist population-based evolutionary algorithm for both the single-source and all-pairs shortest path problems.

This chapter is structured as follows: The level-based analysis will be briefly explained in Section 6.3. In Section 6.4, the algorithm will be explained component by component and the rationale behind the selection of components will be related to the conditions of the level-based theorem. At the end of Section 6.4, the runtime will be proven using a corollary to the level-based theorem and the last section will draw the conclusions.

6.2 The Shortest Path Problem

An instance of a shortest path problem is an input graph $G = (V, E)$, a set $S \subseteq V^2$ of vertex pairs to be connected, and a weight function

for the edges $c(u, v)$ which is defined for each pair of vertices $u \in V$ and $v \in V \setminus \{u\}$. The objective is to find a *path* (sequence of edges) $P^{u,v}$ that connects each vertex pair $(u, v) \in S$ while minimising the total weight $\sum_{(u,v)\in S} \sum_{(w,z)\in P^{u,v}} c(w, z)$. The Single Source Shortest Paths Problem(SSSP) and the All-Pairs Shortest Paths Problem (APSP) are the most studied versions of the problem in the context of EAs. In SSSP, the set of vertex pairs to be connected is $S = \bigcup_{v\in V\setminus\{s\}}(s, v)$ for a source vertex $s \in V$ which is given as part of the input. For the all-pairs version, $S = \{(u, v) \in V^2 | u \neq v\}$, which means that all the vertex pairs should be connected.

We choose the shortest path problem, not only because it is a classical graph problem, but also because its different variations (all-pairs and single source) have been analysed in the context of evolutionary algorithms. The first result for the single-source shortest path problem in the literature was presented by Scharnow et al [77]. The prior analysis was later ramified to cover different evolutionary algorithms and variations of the shortest path problem [3, 26, 23, 27, 47]. A detailed analysis of prior work on the runtime analysis of evolutionary algorithms in the context of shortest path problems is provided in Section 3.3

However certain related problems remain open. Since the previous results were dependent on the monotonic increase in objective function, the performance of non-elitist algorithms with stochastic selection was unknown. While in [23], [27], and [26], a population-based algorithm with a crossover operator was used, neither the population nor the crossover operator was conventional in the sense that the population consisted of a single complete solution divided into independent components and the crossover operator recombined these components to create a new offspring. Therefore, the performance of an algorithm with a "population" – in the sense that a set of self-contained solutions and a crossover operator that recombines information from multiple solutions – has never been analysed in terms of running time.

6.3 Method: Level-Based Analysis

The level-based theorem (Chapter 5) considers the progress of a search process through a given partitioning of the search space, $\mathcal{X}$ with the optimal solutions set $\mathcal{X}^* \subset \mathcal{X}$, into m level sets $(\mathcal{X}, A_2, \ldots, A_{m-1}, \mathcal{X}^*)$ which

are nested in the sense that $\mathcal{X} \supset A_2 \supset A_3 \supset \ldots \supset A_{m-1} \supset \mathcal{X}^*$. The theorem is valid for all population-based algorithms which independently sample each individual in the offspring population from the same distribution.

Theorem 20 considers population-based algorithms at a very abstract level outlined in Algorithm 7 in Chapter 5, such that all the components like fitness evaluations, variation and selection operators are replaced by a single distribution $D(P_t)$ over the solution space $\mathcal{X}$. As shown in earlier work leading to the level-based theorem [63], the identical independent sampling of solutions from this distribution $D(P_t)$ allows the use of concentration of measure results from probability theory.

Algorithm 8 Non-elitist Genetic Algorithm

```
Initialize P_0
t := 0
while Termination conditions are not met do
  for i = 1 to λ do
    p_1 ~ selection(P_t)
    p_2 ~ selection(P_t)
    P_{t+1}(i) := mutation(crossover(p_1, p_2))
  end for
  t := t + 1
end while
```

In order to simplify the use of the theorem, we provided Corollary 21 in Chapter 5 that translates the level-based theorem to the context of genetic algorithms. Since the scope is narrowed, the more general distribution over the search space $D(P_t)$ is replaced by crossover and mutation operators but the details of these operators were not specified so that any algorithm in the form of Algorithm 8 can be analysed with Corollary 21.

The conditions (C1) and (C4) on the mutation and selection operators satisfy the first condition (G1) in the main theorem as our analysis takes only the improvements made by the mutation operator into account. The fourth condition on the selection mechanism implies that having a sufficient number of solutions at level j leads to a high probability of one of them being selected for mutation. A similar breakdown is possible for conditions (C2), (C3) and (C4) since they together imply that the second condition (G2) in the main theorem will be satisfied. The lower bound ε_1 is the probability that the crossover operator produces an offspring solution in a level which is strictly higher than its worst par-

ent. The lower bound p_0 is the probability of not worsening the solution quality. The final condition gives the required population size similarly to the main theorem.

6.4 Design of a Genetic Algorithm

The corollary to the level-based theorem for genetic algorithms produces an upper bound on the expected runtime (Algorithm 8) if the algorithm satisfies certain requirements when optimising a problem. However, when an arbitrary pair of algorithm and problem is analysed, often some of these conflicting requirements are not satisfied. Here we will show that when the theory is leading the design of the algorithm, it is easy to meet the requirements while keeping some flexibility for the algorithm.

In order to apply the corollary, we have to pick a solution representation, a level structure for the search space, a fitness evaluation method, a selection operator, and a variation operator. As a general rule, we aim to keep the term $\sum_{i=1}^{m} 1/s_j$ in the expected optimisation time as small as possible when establishing the level structure. The levels should be few in numbers and close enough in terms of improvement probability. Furthermore, two probabilities are significant for the performance of genetic algorithms: the probability p_0 that the output of the mutation operator will preserve or improve the level of the parent solution (C2), and the probability ε_1 that the crossover operator will generate a solution at a higher level than its worst parent (C3).

While we propose operators for each role in Algorithm 8, the algorithm will still have some flexibility. Because when the goal is an asymptotic runtime result for the algorithm then Corollary 21 only requires asymptotic bounds on the parameters $\{s_j\}_{j\in[m]}$, p_0, and ε_1. This allows the algorithm designer to hybridise the necessary operators proposed in this section with any other operator with constant probability (i.e. using the necessary operator with probability p and the other operator with $1-p$ for any constant $p \in (0,1)$ without changing the upper bound on the expected runtime asymptotically. Such a hybridisation can be used to tackle extraordinary situations in which the main operator has a very small probability of improving the current solution.

In this section we will first design algorithms for the single-source version of the shortest path problem and later discuss the necessary al-

teration for tackling the all-pairs version.

6.4.1 Representation of Solutions

We represent a candidate solution for the single-source shortest path problem as $n-1$ sequences composed of elements of $V \setminus \{s\}$. In the literature mentioned in Section 3.2 different representations that use preceding vertices and sets of edges were used. In our representation which is a sequence of vertices, the source and destination vertices are omitted. A path from the source vertex s to a target vertex v_u that visits vertex v_k just after the source and vertex v_r just before the destination is represented as $P^u = (v_k, \ldots, v_r)$.

We can initialise the population by setting each path $P^i := \emptyset$ which means the only edge in the path is (s, v_i). This initial solution is not only easy to build but also helps skipping the time until the algorithm finds the shortest paths with single edges. Due to the requirements of the theorem we will still need to consider the probabilities of obtaining these single-edge paths in case we lose them because of non-elitist selection operators. Any other initialisation procedure can replace the suggested one as long as it consists of feasible solutions.

6.4.2 The Objective Function and Level Structure

The single-source shortest path problem has a natural objective function which simply sums the weights of all the edges in all the paths. However, the level structure used in the theorem do not have to correlate with the objective function values and a more complex objective function that will guide the algorithm can be designed without further complicating the runtime analysis.

The first step is to convert the sequence of vertices to a set of edges. This conversion is straight forward, for the sequence $P^j = (v_{\pi(1)}, v_{\pi(2)}, v_{\pi(3)}, \ldots)$ the edge set is $E^j = \{(s, v_{\pi(1)}), (v_{\pi(1)}, v_{\pi(2)}), \ldots\}$ and the total weight is $W^j = \sum_{(u,v)\in E^j} c(u, v)$.

If we were able to check whether W^j is the minimum possible value, we could have used the number of correct W^j as an objective function. However since we do not have access to the optimal solution during the run of the algorithm such an objective function would not be possible. Instead whenever we want to compare two solutions we will first sort

W in ascending order and then compare the sorted vectors starting from the first (smallest) value. We will refer to the second smallest values only if the first ones are equal and to the third value only if the first and the second values are the equal, then later values until the tie is broken. This objective function, although complicated, requires only $n \log n$ basic operations to sort the vector and it will allow the genetic algorithm to simulate the well known Dijkstra's algorithm for the shortest path problem [21]. Dijkstra's algorithm fixes the smallest shortest path in each iteration and with the help of this objective function we will force the population to fix the shortest paths in the same order. Since the sub-paths of optimal paths are also optimal, we will use the fixed optimal paths to build other paths that requires more edges.

At this point we decide on how we will partition the search space into levels. In order to achieve a good bound we need levels large enough to keep the number of levels small and small enough to ensure that even the worst solutions in any level has a certain probability of being improved to the next one. When shaping the levels of the solution space we will make use of the optimality of sub-paths as we did in the objective function. However, during analysis we can use the information that is not normally available to the algorithm, like the optimal paths in a candidate solution. W.l.o.g. consider a specific optimal solution $P_* = \{P_*^1, P_*^2, \ldots, P_*^{n-1}\}$ and $P_*^\pi = (P_*^{\pi(1)}, P_*^{\pi(2)}, \ldots, P_*^{\pi(n-1)})$ the sorted P_* such that $W_*^{\pi(i)} \leq W_*^{\pi(i+1)}\ \forall i \in [n-1]$. So, we divide the solution space $\mathcal{X}$ into n levels, where for all $j \in \{0, \ldots, n-1\}$, the level A_j consists of all the solutions that contains $\{P_*^{\pi(i)} | 1 \leq i \leq j\}$, the first j shortest paths of the sorted optimal solution P_*^π .

6.4.3 The Mutation Operator

With a fixed level structure, the choice of mutation operator will determine the upgrade probabilities $\{s_j \mid j \in [n]\}$ required by condition (C1) and p_0 required by condition (C2), the probability that the mutation operator increases or maintains the quality of the input solution.

Consider the mutation operator presented in Operator 9, which as a local operation selects a path x^i, uniformly at random, to be modified either by removing all the vertices in the path with probability $1/n$ or with probability $1 - 1/n$ replaced by another randomly selected path x^k

Operator 9 mutation(x) (SSSP)

Input: A solution x consisting of $n-1$ sequences of vertices $x^1, x^2, \ldots, x^{n-1}$
Pick $s \sim \text{Pois}(1)$
for $\ell = 1$ to s **do**
  For $i \sim \text{Unif}([n-1])$
  if with probability $1/n$ **then**
    Set $x^i := \emptyset$
  else
    Set $x^i := (x^k, v_k)$ for $k \sim \text{Unif}([n-1] \setminus \{i\})$
  end if
end for
Return: **x**

with the destination vertex v_k appended at the end. First of all, since the number of local operations S is distributed according to $\text{Pois}(1)$, the probability that S is equal to zero is $1/e$. Since $S = 0$ implies that the solution is not changed at all, we have $p_0 \geq 1/e$, which satisfies (C2). For condition (C1), we need to prove that this operator can improve any solution from level j to level $j+1$ at least with some probability s_j. Note that if a solution is in level j, the smallest j paths in the optimal solution are already set correctly. For improving to level $(j+1)$ it is sufficient to set the $(j+1)$th smallest optimal path, $P_*^{\pi(j+1)}$, correctly, without tempering with any of the first correctly set j paths. Let's assume that this path has more than one edge, i.e. it visits at least one more vertex v_r before the destination of $P_*^{\pi(j+1)}$. Consider the path to vertex v_r in the optimal solution, which is shorter than $P_*^{\pi(j+1)}$ because the edge weights are positive. Then if our solution is on level j , it has the optimal sub-path up to any vertex v_r that needs to be visited before the destination of $P_*^{\pi(j+1)}$. The mutation operator picks $S = 1$ with probability $1/e$, then with probability $1/(n-1)$, $x^{\pi(j+1)}$ in the current solution is picked for mutation. With probability $1 - 1/n$, $x^{\pi(j+1)}$ will not be erased and the correct subpath of $P_*^{\pi(j+1)}$ can be chosen to replace $x^{\pi(j+1)}$ with probability $1/(n-2)$. In total with probability $\frac{1}{en(n-2)}$, the $\pi(j+1)$th path in the offspring solution $mut(x)^{\pi(j+1)}$ will be set to the optimal path $P_*^{\pi(j+1)}$ and no other modification will be made. Then, the offspring solution $mut(x)$ will be in A_{j+1}. We can now bound $s_j \geq \frac{1}{en(n-2)} = \Omega(\frac{1}{n^2})$.

If $P_*^{\pi(j+1)}$ has a single edge $(s, v_{\pi(j+1)})$ the mutation operator has to remove all the vertices in $x^{\pi(j+1)}$. If $x^{\pi(j+1)}$ is selected for mutation with probability $1/(n-1)$ then with probability $1/n$ the mutation operator

removes all the vertices in the path. In total the probability that $mut(x) \in A_{j+1}$ is similarly $s_j \geq \frac{1}{en(n-1)} = \Omega(\frac{1}{n^2})$

6.4.4 The Crossover Operator

It is shown in some of the earlier works on EAs and the all-pairs shortest path problem that the crossover operator helps creating paths that involve more edges and allows a multiplicative growth in the number of optimal paths discovered. However, in the context of Corollary 21, the contribution from the crossover operator is to ensure that individuals with average quality in the solution will be improved if they are recombined with one of the high quality solutions. The lower bound ε_1 in the (C3) of Corollary 21 is the guiding value when deciding how our crossover will work. Consider two solutions $x^1 \in A_j$ and $x^2 \in A_{j+1}$, i.e., x^1 and x^2 share the j shortest paths in the optimal solution and x^2 alone has the $(j+1)$-th shortest path. Our condition implies that the offspring solution should have all these j paths and the $(j+1)$-th path, so that it will be on a level strictly higher than its worse parent, x^1. Therefore, the crossover operator should keep all the shared j paths and pick the $(j+1)$ from x^2 with a reasonable probability. This cannot be accomplished by the crossover operators that concatenates the paths as in [24] since even though they allow large improvements they also have a large probability of worsening the solution. A simpler crossover on the other hand, which picks each path from either one of the parents with probability $1/2$ will result in a constant ε_1. This is because the shared paths will be copied to the offspring and the extra path that makes the difference between the parents will be selected from the better parent with probability $\epsilon_1 = 1/2$.

Operator 10 crossover(x^1, x^2)

Input: Two solutions x and y each consisting of k sequences of vertices
for $i = 1$ to k **do**
  if with probability $1/2$ **then**
    Set $z^i := x^i$
  else
    Set $z^i := y^i$
  end if
end for
Return: z

6.4.5 Other Parameter Settings

In the context of Algorithm 8, fixing the mutation and the crossover operators mostly concludes the decisions for the algorithm. The remaining parameters like the population size or the parameters of the selection mechanism can be directly set according to the result of Corollary 21 and Lemma 22.

The selection mechanism for our algorithm can be chosen among some of the widely used non-elitist selection mechanisms, such as k-tournament selection, exponential ranking selection, and (μ, λ)-selection. The required parameters for each of these selection mechanisms are given in Lemma 22 and since the p_0 and ε_1 are both constant, we can conclude that the required parameters k, η and μ/λ are also constant.

By condition (C5), the population size must satisfy $\lambda \geq \frac{2}{a} \ln\left(\frac{32mp_0}{(\delta\gamma_0)^2 cs_*\psi}\right)$. This means that the minimum population size logarithmically depends on the inverse of smallest improvement probability, the number of levels m, p_0 and ε_1. Since the latter two are constant and the smallest improvement probability is in $\Omega(1/n^2)$ we conclude that a population size of $\lambda \in \mathcal{O}(\log n)$ satisfies condition (C5).

6.4.6 All-Pairs Shortest Path Problem

In this section we will adapt the genetic algorithm we designed for SSSP to the all-pairs version of the shortest path problem. The obvious change is in the problem representation necessary for SSSP's algorithm to be applied to the APSP. The complete solution of APSP problem consist of $n(n-1)$ sequence of vertices between $n(n-1)$ pairs of vertices. The source and the destination vertices are omitted from the sequence and the sequence of vertices are converted to a set of edges in a similar fashion as in SSSP. For the sequence $P^{ij} = (v_{\pi(1)}, v_{\pi(2)}, v_{\pi(3)}, \ldots)$ of length ℓ, the edge set is $E^{ij} = \{(v_i, v_{\pi(1)}), (v_{\pi(1)}, v_{\pi(2)}), \ldots, (v_{\pi(\ell)}, v_j)$ and the total weight is $W^{ij} = \sum_{(u,v)\in E^{ij}} c(u,v)$. Similar changes in level structure of the search space necessary for the purpose of analysis. We divide the solution space $\mathcal{X}$ into $m = n(n-1)+1$ levels, where for all $j \in \{0, \ldots, m\}$, the level A_j consists of all the solutions that contains the smallest j shortest paths of the optimal solution.

Since the solution representation is changed, some modifications are necessary for the mutation operator as well.

Operator 11 mutation(x) (APSP)

Input: A solution x consisting of $n(n-1)$ sequences of vertices $x^{1,2}, x^{1,3}, \ldots, x^{1,(n)}, \ldots, x^{i1}, x^{i2}, \ldots, x^{i,(i-1)}, x^{i,(i+1)}, \ldots x^{n,(n-1)}$
Pick $s \sim \text{Pois}(1)$
for $\ell = 1$ to s **do**
  For $i \sim \text{Unif}([n])$ and $j \sim \text{Unif}([n] \setminus \{i\})$
  if with probability $1/n$ **then**
    Set $x^i j := \emptyset$
  else
    Set $x^i j := (x^{ik}, v_k)$ for $k \sim \text{Unif}([n] \setminus \{i\})$
  end if
end for
Return: **x**

The most important modification to the mutation operator is that it now picks the path that will replace the path x^{ij} among the paths in the form x^{ik} that share the destination vertex v_i. For APSP, the lower bound on the improvement probabilities s_i are $\Omega(1/n^3)$ since the correct mutation step involves picking the right path among $n(n-1)$ alternatives (rather than n) and picking the correct vertex v_k to append among $\mathcal{O}(n)$ alternatives. Since the number of local operations is $Pois(1)$, the bound on p_0 is still $1/e$.

The solution comparison procedure and crossover operators are identical to the ones used for SSSP. The crossover operator maintains the same values for parameters ϵ_1.

6.5 Expected Running Time

Here, as promised above we provide the runtime of our algorithm on the single-source shortest path problem.

Theorem 26. *The non-elitist GA in Algorithm 8 using the crossover operator given in Operator 10, and the mutation operator given in Operator 9, and either k-tournament selection with $k \geq 8e(1+\delta)$, or (μ, λ)-selection with $\lambda/\mu \geq 2e(1+\delta)$ or exponential ranking selection with $\eta \geq 8e(1+\delta)$, for a constant $\delta > 0$, and population size $\lambda \geq c \ln n$ for some constant $c > 0$, has expected runtime $O(n^3)$ on the single-source shortest path problem.*

Proof. We show that the algorithm satisfies the conditions of Corollary 21. Condition (C1) is satisfied for $s_* \in \Omega(1/n^2)$. Condition (C2) is satisfied for $p_0 \geq 1/e$ by the Operator 9. Condition (C3) is satisfied for

$\varepsilon_1 \geq 1/2$ by the Operator 10. Condition (C4) is satisfied for some constant γ_0 by the given parameters in Theorem 26 due to Lemma 22. Finally, condition (C5) is satisfied since $m = n$, $s_* \in \Omega(1/n^2)$, $p_o \in \mathcal{O}(1)$ and $\varepsilon_1 \in \mathcal{O}(1)$. The expected runtime $\mathcal{O}(n^3)$ is obtained by plugging in the parameters m , s_j, p_0 and ε_1 to the expression in the Corollary 21. □

The same result also holds for the all-pairs version with the modification to improvement probabilities and the number of levels.

Theorem 27. *The non-elitist GA in Algorithm 8 using the crossover operator given in Operator 10, and the mutation operator given in Operator 11, and either k-tournament selection with $k \geq 8e(1+\delta)$, or (μ, λ)-selection with $\lambda/\mu \geq 2e(1+\delta)$ or exponential ranking selection with $\eta \geq 8e(1+\delta)$, for a constant $\delta > 0$, and population size $\lambda \geq c \ln n$ for some constant $c > 0$, has expected runtime $O(n^5)$ on the all-pairs shortest path problem.*

Proof. We show that the algorithm satisfies the conditions of Corollary 21. Condition (C1) is satisfied for $s_* \in \Omega(1/n^3)$. Condition (C2) is satisfied for $p_0 \geq 1/e$ by the Operator 9. Condition (C3) is satisfied for $\varepsilon_1 \geq 1/2$ by the Operator 10. Condition (C4) is satisfied for some constant γ_0 by the given parameters in Theorem 27 due to Lemma 22. Finally, condition (C5) is satisfied since $m = n(n-1)+1$, $s_* \in \Omega(1/n^3)$, $p_o \in \mathcal{O}(1)$ and $\varepsilon_1 \in \mathcal{O}(1)$. The expected runtime $\mathcal{O}(n^5)$ is obtained by plugging in the parameters m , s_j, p_0 and ε_1 to the expression in the Corollary 21. □

6.6 Discussion

We may want to compare our result on with the performance of EA without crossover since the elementary crossover operator we use is fundamentally different than the crossover operators used in the prior literature. The upper bound of $\mathcal{O}(n^3)$ is close to the performance of elitist EAs. Still, we need to remark that it does not reflect the exact optimisation time in terms of elementary operation. There is a missing $\Theta(\ln(n))$ factor when we only count the number of function evaluations since the comparison mechanism relies on sorting the paths in the solution. Since this sorting procedure is not used in the elitist algorithms that solves the problem, we can conclude that our upper bound is worse by a logarithmic factor.

Even with the alteration to the algorithm, the discrepancy between the upper bounds on expected runtime of the elitist EA and non-elitist GA is larger for APSP than it is for SSSP. The expected runtime is $\Theta(n^4)$ for EA's without crossover on APSP [23]. If we were to compare this result with the tight runtime of $\Theta(n^3)$ for SSSP, we see a multiplicative factor of n which is admissible since the all-pairs shortest path problem can be divided into n SSSP problems with n different source vertices. However, we will observe a looser upper bound of $\mathcal{O}(n^5)$ for APSP while the upper bound we provided for SSSP is $\mathcal{O}(n^3)$.

Prior evolutionary algorithms that solve APSP, alter only a single path between two objective function evaluations. This allows an objective function to distinguish between improvements and deterioration even when the objective function aggregates the weights of all paths. However, when two arbitrary solutions are compared that differ in more than one path, the aggregate objective function value can be deceptive. This precision loss is critical for the non-elitist algorithm since unless higher level solutions are favoured by the objective function the progress over levels cannot be sustained.

The underlying idea of the $\mathcal{O}(n^4)$ upper bound for the elitist EA is that the mutation operator can improve any path at most n times before that path becomes optimal[23]. A corresponding level structure for the non-elitist case would be the total number of improvements required for reaching the optimal solution. With this level structure it is not guaranteed that a solution at a better level will always be favoured by the objective function. A similar obstacle was present for the level structure used for SSSP. However, for the proposed level structure of SSSP, being on level j required having "all" the j optimal paths with smallest total weights, not having a total of j optimal paths. The comparison procedure which sorts the paths in order of increasing total weight and comparing the shortest paths first guaranteed that solutions of higher level are preferred. For the aggregate number of necessary improvements how such a method might work is not apparent. So we restrict ourselves to the same level structure used for SSSP for the analysis of APSP as well, even though the number of levels is liable for the increase in the upper bound on expected optimisation time.

6.7 Summary

In this chapter we reversed conventional runtime analysis and designed an algorithm which fits the analytical tool, in this case level-based analysis presented in Chapter 5. By doing so, we obtained a rigorously proven performance result by simply plugging in the parameters of the resulting algorithm. Tracing the requirements of the theoretical results guided the design of a provably efficient algorithm without going through tedious analysis. Moreover, we showed that using a high level runtime tool as a design guide can ease the experimental burden, not only for evaluating performance but also for setting parameters since the theorem provides the required population size and specific parameters for selection operators.

We saw that the essential design decisions were the addition of a subroutine that allows bulk deletion of vertices in a path and the choice of a simple and stable crossover. We exploited the strength of our analysis tool and incorporated the problem specific information to our objective function that allowed us to simulate a well-known deterministic algorithm without making the runtime analysis any harder. We also kept some degrees of freedom for the algorithm by allowing the use of hybridised operator and a variety of selection mechanisms.

On the theoretical side, this chapter considers a non-elitist population-based algorithm, which was hard to analyse before the introduction of the level-based theorem and previously not considered for many popular problems, including the single-source and all-pairs shortest path problems. Moreover, we provided the first results that use a binary crossover operator on shortest path problems, which is an important result since the all-pairs version of this problem is known to be the first non-artificial problem that asymptotically benefited from the use of a crossover operator.

Chapter 7

Conclusions

Evolutionary algorithms (EAs) are general purpose optimisation methods which can be applied to a wide range of problems. The performance of EAs on a particular problem can be measured with respect to the expected computational resources necessary for solving the problem. In particular, runtime analyses of EAs aim at obtaining rigorously proven bounds for these performance measures. Even slight changes in the problem definition or the parameters of the algorithm can affect the performance of an EA. One such important aspect is how the quality of candidate solutions are evaluated, *i.e.*, the fitness function.

This thesis investigates the use of fitness evaluation methods with the aim of showing how the performance of EAs can improve when more problem knowledge is incorporated in the fitness evaluation method. The runtime of evolutionary algorithms with different fitness evaluation methods has been analysed for hard combinatorial optimisation problems. We have devised a new analytical tool to understand the progress of search processes without referring to the progress in the objective function value. This tool not only allows analysis of a wide range of non-elitist population-based EAs with sophisticated variation operators but it can also allow informed design of EAs with more involved solution comparison methods than a standard fitness function.

The comparison of different fitness evaluation methods resulted in the first runtime analysis of evolutionary algorithms on bi-level optimisation problems. In particular, we have shown that our proposed evolutionary algorithms solve the GMSTP and GTSP problems in polynomial time if the number of clusters is kept constant (*i.e.* it is a fixed parameter). Bi-level problems have a structure clearly separated into lower and

upper-level optimisation problems which hints at where a fitness evaluation method might fit. The effects of fitness evaluation methods on the performance of the resulting algorithm are easily analysed. Unlike bi-level problems which have conflicting objective functions for lower and upper levels (*e.g.* the toll setting problem[57]), GMSTP and GTSP are not inherently bi-level. Here, the bi-level structure divides the problem into two parts which are solved by two different algorithms. This approach allowed us to illustrate how the responsibility of evaluating the fitness of an upper-level search point can be delegated by the EA to another subroutine which optimises the lower level problem. As the solution of the lower level optimisation problem provides the fitness of the higher level solutions, bi-level optimisation *per se* is a particular fitness evaluation strategy. Our results indicate that representing hard combinatorial optimisation problems as bi-level optimisation problems can yield efficient evolutionary algorithms in the problem size while the problem hardness is isolated in the fixed parameter. Using parameterised computational complexity analysis of evolutionary algorithms for the generalized minimum spanning tree problem, we have examined two representations for the upper-level problem and used problem-specific algorithms for the lower-level problem. Our results show that the so-called *Global Structure Representation* leads to fixed parameter evolutionary algorithms. By presenting hard instances for each of the two approaches, we have proven matching lower bounds on their parameterised runtime. Furthermore, we have shown that the two representations for the generalized minimum spanning tree problem are complementary by proving that each algorithm is highly efficient on the hard instance of the other algorithm. A significant result of the analysis was that not all divisions between upper and lower level problems have the same performance. In particular, different asymptotic runtimes for different bi-level structures for GMST emphasised the importance of which task is delegated to the fitness evaluation. Later, these results are extended to the GTSP problem. The parameterised analysis used the number of clusters as main parameter in both problems and this allowed the transfer of results from the GMST problem to the GTSP problem. By extending the results to the GTSP problem and providing the necessary characteristics of the bi-level structure for further extensions,a general framework for the use of fitness evaluation methods in hard combinatorial optimisation problems

is established.

The obvious future work in this direction is to find bi-level representations of other hard combinatorial optimisation problems where the size of the upper level solution depends on a parameter which tends to be relatively small in real world instances. This would allow the identification of a real-world optimisation problem for which EAs are very efficient. A promising direction might be to analyse inherently bi-level problems with conflicting objective functions since the division into upper and lower levels would be pre-specified and it would be easy to verify whether our presented framework applies. Overall, even for problems which are not inherently bi-level, dividing the problem into multiple levels can be a practically viable strategy when tackling hard problems with evolutionary algorithms.

After initiating our research with problem-specific results, we turned our attention to the available high level tools for runtime analysis of evolutionary algorithms. Tools that exploit the frequent and incremental improvements made by evolutionary algorithms, e.g. artificial fitness levels method, were among the most commonly applied. In order to adapt these tools to arbitrary fitness evaluation methods, "the level-based theorem" is proposed. This new theorem proves upper bounds on the runtime of non-elitist evolutionary algorithms which independently sample each individual in the next generation from the same probability distribution over the search space. Unlike the classical fitness level theorem, the level-based theorem allows any nested partitioning of the search space, and not only fitness-based partitions. The freedom to choose a partitioning independent from the objective function allows derivation of the runtimes of evolutionary algorithms with more complex fitness functions. Moreover, the level-based theorem covers a large class of algorithms since the requirements are defined for the characterising probability distribution of the sampling process. A corollary to the theorem that bounds the runtime of genetic algorithms with independent sampling is provided. Given an appropriate balance between selection and variation operators, we have shown that GAs optimise standard benchmark functions, as well as combinatorial optimisation problems, efficiently. Our results allow the analysis of a wide array of settings for which no prior mathematical tools were available. Moreover, the straightforward to verify conditions of the theorem allow earlier proofs in the literature to be

simplified. The core of the future work that stems from the introduction of level-based analysis is its application on various non-elitist algorithms and problems. However, we should also emphasize the potential improvements on the theorem itself. The most significant shortcoming of the corollary for genetic algorithms is that it fails to fully take into account any improvements made by the recombination operators. Currently, the theorem only accounts for the crossover operators' role in duplicating good individuals, and does not take into account other benefits of crossover. In order to asses the contribution of recombination operators to the improvement of solutions, the first obstacle is tracking the diversity of the population. By itself, bounding the diversity from below in a setting with independent sampling is a challenge. Incorporating the diversity of the population to the distance function that represents the state of the population is a further complication that needs to be addressed.

The most significant application of level-based analysis in this thesis is the design of a genetic algorithm for the shortest path problem with the guidance of the conditions of the level-based theorem. These conditions are translated into design decisions leading to an algorithm with a proven upper bound on its runtime. For the runtime analysis, we consider a non-elitist population-based EA, which was hard to analyse before the introduction of the level-based theorem. Moreover, we provided the first results for EAs that use a binary crossover operator on shortest path problems. This is an important result since the all-pairs version of this problem is known to be the first non-artificial problem that asymptotically benefited from the use of a crossover operator. We saw that the essential design decisions were the addition of a subroutine that allows bulk deletion of vertices in a path and the choice of a simple and stable crossover. Overall, our research shows how analytical tools can also be used as design guidelines for provably efficient algorithms. Besides its contribution as a new design framework which exploits the theory, the main contribution of this work to the aim of the thesis is the use of a highly tailored fitness evaluation method. The proposed algorithm does not use an explicit fitness function which maps a solution to a real value. Instead it operates as a part of the selection mechanism and compares solutions according to their contribution to the optimisation process. This solution comparison method is inspired by the famous Di-

jkstra algorithm for the shortest path problem which corrects the paths in the shortest path tree starting from the ones with the minimum weight. We transferred this focus on paths with smaller weights to our solution comparison method so that the algorithm would mimic Dijkstra's algorithm. In future work, this approach can be generalised to "force" evolutionary algorithms to mimic dynamic programming. In a more general direction though, the use of level-based analysis to design fitness evaluation methods should be extended to other more complex optimisation problems.

Bibliography

[1] Karl A Abrahamson, Rodney G Downey, and Michael R Fellows. Fixed-parameter tractability and completeness IV: on completeness for w [P] and PSPACE analogues. *Annals of Pure and Applied Logic*, 73(3):235–276, 1995.

[2] Anne Auger and Benjamin Doerr. *Theory of Randomized Search Heuristics: Foundations and Recent Developments*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2011.

[3] Surender Baswana, Somenath Biswas, Benjamin Doerr, Tobias Friedrich, Piyush P. Kurur, and Frank Neumann. Computing single source shortest paths using single-objective fitness functions. In *Proceedings of the International Workshop on Foundations of Genetic Algorithms (FOGA '09)*, page 59–66. ACM Press, 2009.

[4] Tianshi Chen, Jun He, Guangzhong Sun, Guoliang Chen, and Xin 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: Cybernetics*, 39(5):1092–1106, 2009.

[5] Tianshi Chen, Per Kristian Lehre, Ke Tang, and Xin Yao. When is an estimation of distribution algorithm better than an evolutionary algorithm? In *Proceedings of IEEE Congress on Evolutionary Computation (CEC 2009)*, page 1470–1477. IEEE, 2009.

[6] Benoît Colson, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. *Annals of Operations Research*, 153(1):235–256, 2007.

[7] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC '71)*, page 151–158, New York, NY, USA, 1971. ACM.

[8] Dogan Corus, Duc-Cuong Dang, Anton V. Eremeev, and Per Kristian Lehre. Level-based analysis of genetic algorithms and other search processes. In *Proceedings of the 13th International Conference on Parallel Problem Solving from Nature (PPSN 2014)*, page 912–921, Ljubljana, Slovenia, 2014. Springer International Publishing.

[9] Dogan Corus, Per Kristian Lehre, and Frank Neumann. The generalized minimum spanning tree problem: a parameterized complexity analysis of bi-level optimisation. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO '13)*, page 519–526, New York, NY, USA, 2013. ACM.

[10] Dogan Corus, Per Kristian Lehre, Frank Neumann, and Mojgan Pourhassan. A parameterised complexity analysis of bi-level optimisation with evolutionary algorithms. *Evolutionary Computation*, 24(1), 2015.

[11] Duc-Cuong Dang, Thomas Jansen, and Per Kristian Lehre. Populations can be essential in dynamic optimisation. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO '15)*, pages 1407–1414, New York, NY, USA, 2015. ACM.

[12] Duc-Cuong Dang and Per Kristian Lehre. Refined upper bounds on the expected runtime of non-elitist populations from fitness-levels. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO '14)*, page 1367–1374, 2014.

[13] Duc-Cuong Dang and Per Kristian Lehre. Efficient optimisation of noisy fitness functions with population-based evolutionary algorithms. In *Proceedings of the International Workshop on Foundations of Genetic Algorithms (FOGA '15)*, pages 62–68, New York, NY, USA, 2015. ACM.

[14] Duc-Cuong Dang and Per Kristian Lehre. Simplified runtime analysis of estimation of distribution algorithms. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO '15)*, pages 513–518, New York, NY, USA, 2015. ACM.

[15] Duc-Cuong Dang and Per Kristian Lehre. Self-adaptation of mutation rates in non-elitist populations. In *Proceedings of the Inter-*

national Conference on Parallel Problem Solving from Nature (PPSN 20016), LNCS, to appear 2016.

[16] Charles Darwin. *On the origin of species*. John Murray, 1859.

[17] Kenneth A De Jong. *Evolutionary computation: a unified approach*. MIT press, 2006.

[18] Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons, 2001.

[19] Kalyanmoy Deb and Ankur Sinha. Solving bilevel multi-objective optimization problems using evolutionary algorithms. In *Evolutionary Multi-Criterion Optimization*, page 110–124. Springer, 2009.

[20] Kalyanmoy Deb and Ankur Sinha. An efficient and accurate solution methodology for bilevel multi-objective programming problems using a hybrid evolutionary-local-search algorithm. *Evolutionary computation*, 18(3):403–449, 2010.

[21] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[22] Benjamin Doerr, Edda Happ, and Christian 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 2007)*, pages 1890–1895, 2007.

[23] Benjamin Doerr, Edda Happ, and Christian Klein. Crossover can provably be useful in evolutionary computation. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO '08)*, page 539–546, New York, NY, USA, 2008. ACM.

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

[25] Benjamin Doerr and Daniel Johannsen. Edge-based representation beats vertex-based representation in shortest path problems. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO '10)*, page 759–766. ACM, 2010.

[26] Benjamin Doerr, Daniel Johannsen, and Timo Kötzing. More effective crossover operators for the all-pairs shortest path problem. In *Proceedings of the International Conference on Parallel Problem Solving from Nature (PPSN 2010)*, page 184–193. Springer, 2010.

[27] Benjamin Doerr and Madeleine Theile. Improved analysis methods for crossover-based algorithms. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO '09)*, page 247–254, New York, NY, USA, 2009. ACM.

[28] Rod G Downey and Michael R Fellows. *Parameterized complexity*. Springer, 1999.

[29] Stefan Droste, Thomas Jansen, and Ingo Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276(1-2):51–81, April 2002.

[30] Stefan Droste, Thomas Jansen, and Ingo Wegener. Upper and lower bounds for randomized search heuristics in Black-Box optimization. *Theory of Computing Systems*, 39(4):525–544, 2006.

[31] Devdatt P Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.

[32] Agoston E. Eiben and James E. Smith. *Introduction to evolutionary computing*. Springer, 2008.

[33] Anton V Eremeev. Modeling and analysis of genetic algorithm with tournament selection. In *Artificial Evolution*, page 84–95. Springer, 2000.

[34] Matteo Fischetti, Juan Jose Salazar Gonzalez, and Paolo Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45(3):378–394, 1997.

[35] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[36] Wanru Gao, Tobias Friedrich, and Frank Neumann. Fixed-parameter single objective search heuristics for minimum vertex cover. In *Parallel Problem Solving from Nature - PPSN XII*, pages 740–750. Springer Berlin Heidelberg, 2016.

[37] Oliver Giel and Ingo Wegener. Evolutionary algorithms and the maximum matching problem. In *Proceedings of the 20th Symposium on Theoretical Aspects of Computer Science (STACS '03)*, volume 2607, page 415–426, 2003.

[38] John H. Gillespie. Molecular evolution over the mutational landscape. *Evolution*, 38(5):1116–1129, 1984.

[39] David E Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Longman, 1989.

[40] David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, page 69–93. Morgan Kaufmann, 1991.

[41] Gregory Gutin and Abraham P Punnen. *The traveling salesman problem and its variations*, volume 12. Springer Science & Business Media, 2002.

[42] Bruce Hajek. Hitting-Time and Occupation-Time bounds implied by drift analysis with applications. *Advances in Applied Probability*, 14(3):502, September 1982.

[43] Jun He and Xin Yao. Drift analysis and average time complexity of evolutionary algorithms. *Artificial Intelligence*, 127(1):57–85, 2001.

[44] Jun He and Xin Yao. From an individual to a population: An analysis of the first hitting time of Population-Based evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):495–511, 2002.

[45] Jun He and Xin Yao. Towards an analytic framework for analysing the computation time of evolutionary algorithms. *Artificial Intelligence*, 145(1-2):59–97, 2003.

[46] Jun He and Xin Yao. A study of drift analysis for estimating computation time of evolutionary algorithms. *Natural Computing*, 3(1):21–35, 2004.

[47] Christian Horoba. Analysis of a simple evolutionary algorithm for the multiobjective shortest path problem. In *Proceedings of the International Workshop on Foundations of Genetic Algorithms (FOGA '09)*, pages 113–120, 2009.

[48] Bin Hu and Günther Raidl. An evolutionary algorithm with solution archives and bounding extension for the generalized minimum spanning tree problem. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO '12)*, page 393–400. ACM, 2012.

[49] Bin Hu and Günther R Raidl. An evolutionary algorithm with solution archive for the generalized minimum spanning tree problem. In *Computer Aided Systems Theory–EUROCAST 2011*, page 287–294. Springer, 2012.

[50] Thomas Jansen and Ingo Wegener. On the analysis of evolutionary algorithms—a proof that crossover really can help. In *Proceedings of 7th Annual European Symposium on Algorithms (ESA 1999)*, volume 1643 of *LNCS*, page 700, 1999.

[51] Thomas Jansen and Ingo Wegener. The analysis of evolutionary Algorithms–A proof that crossover really can help. *Algorithmica*, 34(1):47–66, July 2002.

[52] Thomas Jansen and Ingo Wegener. Real royal road functions—where crossover provably is essential. *Discrete Applied Mathematics*, 149(1-3):111–125, August 2005.

[53] John G. Kemeny and James Laurie Snell. *Finite Markov chains*. Undergraduate texts in mathematics. Springer, New York, 1976. Reprint of the 1960 ed. published by Van Nostrand, Princeton, N.J., in the University series in undergraduate mathematics.

[54] Andrew Koh. Solving transportation bi-level programs with differential evolution. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC 2007)*, page 2243–2250, 2007.

[55] Stefan Kratsch, Per Kristian Lehre, Frank Neumann, and Pietro Simone Oliveto. Fixed parameter evolutionary algorithms and maximum leaf spanning trees: A matter of mutation. In *Proceedings of the International Conference on Parallel Problem Solving from Nature (PPSN 2010)*, page 204–213. Springer, 2010.

[56] Stefan Kratsch and Frank Neumann. Fixed-parameter evolutionary algorithms and the vertex cover problem. *Algorithmica*, 65(4):754–771, 2013.

[57] Martine Labbé, Patrice Marcotte, and Gilles Savard. A bilevel model of taxation and its application to optimal highway pricing. *Management Science*, 44(12-part-1):1608–1622, 1998.

[58] François Legillon, Arnaud Liefooghe, and E Talbi. Cobra: A cooperative coevolutionary algorithm for bi-level optimization. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC 2012)*, page 1–8. IEEE, 2012.

[59] Per Kristian Lehre. Negative drift in populations. In *Proceedings of the International Conference on Parallel Problem Solving from Nature (PPSN 2010)*, volume 6238 of *LNCS*, page 244–253. Springer, 2010.

[60] Per Kristian Lehre. Fitness-levels for non-elitist populations. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO '11)*, page 2075–2082, 2011.

[61] Per Kristian Lehre and Xin Yao. Crossover can be constructive when computing unique input output sequences. In *Asia-Pacific Conference on Simulated Evolution and Learning*, pages 595–604. Springer, 2008.

[62] Per Kristian Lehre and Xin Yao. On the impact of the mutation-selection balance on the runtime of evolutionary algorithms. In *Proceedings of the tenth ACM SIGEVO workshop on Foundations of genetic algorithms (FOGA 2009)*, page 47–58, New York, NY, USA, 2009. ACM.

[63] Per Kristian Lehre and Xin Yao. On the impact of Mutation-Selection balance on the runtime of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 16(2):225–241, April 2012.

[64] Orna Lichtenstein and Amir Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '85, page 97–107, New York, NY, USA, 1985. ACM.

[65] Zbigniew Michalewicz. A survey of constraint handling techniques in evolutionary computation methods. *Evolutionary Programming*, 4:135–155, 1995.

[66] Wil Michiels, Emile Aarts, and Jan Korst. *Theoretical aspects of local search*. Springer Science & Business Media, 2007.

[67] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Chapman & Hall/CRC, 2010.

[68] Young-Soo Myung, Chang-Ho Lee, and Dong-Wan Tcha. On the generalized minimum spanning tree problem. *Networks*, 26(4):231–241, 1995.

[69] S. Nallaperuma, A.M. Sutton, and F. Neumann. Fixed-parameter evolutionary algorithms for the Euclidean traveling salesperson problem. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC 2013)*, pages 2037–2044, June 2013.

[70] Frank Neumann and Ingo Wegener. Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO '04)*, volume 3102 of *LNCS*, page 713–724. Springer-Verlag, 2004.

[71] Frank Neumann and Carsten Witt. *Bioinspired Computation in Combinatorial Optimization: Algorithms and Their Computational Complexity*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.

[72] Pietro S. Oliveto and Carsten Witt. Improved runtime analysis of the simple genetic algorithm. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO '13)*, page 1621–1628, 2013.

[73] Tiago Paixão, Golnaz Badkobeh, Nicholas H. Barton, Dogan Corus, Duc-Cuong Dang, Tobias Friedrich, Per Kristian Lehre, Dirk Sudholt, Andrew M. Sutton, and Barbora Trubenova. Toward a unifying framework for evolutionary processes. *Journal of Theoretical Biology*, 2015.

[74] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Dover Publications, 1998.

[75] Petrica Claudiu Pop. *The generalized minimum spanning tree problem*. PhD thesis, Twente University, 2002.

[76] Günter Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE transactions on neural networks*, 5(1):96–101, 1994.

[77] Jens Scharnow, Karsten Tinnefeld, and Ingo Wegener. The analysis of evolutionary algorithms on sorting and shortest paths problems. *Journal of Mathematical Modelling and Algorithms*, 3(4):349–366, 2005.

[78] Dirk Sudholt. A new method for lower bounds on the running time of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 17(3):418–435, June 2013.

[79] Andrew M Sutton, Jareth Day, and Frank Neumann. A parameterized runtime analysis of evolutionary algorithms for MAX-2-SAT. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO '12)*, page 433–440. ACM, 2012.

[80] Andrew M Sutton and Frank Neumann. Parameterized runtime analyses of evolutionary algorithms for the Euclidean traveling salesperson problem. *arXiv preprint arXiv:1207.0578*, 2012.

[81] Andrew M. Sutton, Frank Neumann, and Samadhi Nallaperuma. Parameterized runtime analyses of evolutionary algorithms for the planar Euclidean traveling salesperson problem. *Evolutionary Computation*, 22(4):595–628, 2014.

[82] AndrewM. Sutton and Frank Neumann. A parameterized runtime analysis of simple evolutionary algorithms for makespan scheduling. In *Proceedings of the International Conference on Parallel Problem Solving from Nature (PPSN 2012)*, volume 7491 of *Lecture Notes in Computer Science*, pages 52–61. Springer Berlin Heidelberg, 2012.

[83] Michael D. Vose. *The Simple Genetic Algorithm: Foundations and Theory*. The MIT Press, Cambridge, Mass, August 1999.

[84] Ingo Wegener. Methods for the analysis of evolutionary algorithms on Pseudo-Boolean functions. In *Evolutionary Optimization*, volume 48, pages 349–369. Springer US, 2002.

[85] Michael James Denham White et al. *The chromosomes.* 1937.

[86] Carsten Witt. Runtime analysis of the (μ+1) EA on simple Pseudo-Boolean functions. *Evolutionary Computation*, 14(1):65–86, 2006.

[87] David H Wolpert and William G Macready. No free lunch theorems for optimization. *Optimization*, 1(1):67–82, 1997.

Appendices

The following are basic lemmas that will be used for the proof of Theorem 20.

Theorem 28 (Bernoulli's Inequality). *If $x > -1$ and $n \in \mathbb{N}$, then $(1+x)^n \geq 1 + nx$.*

Lemma 29. *For all $x \geq 0$, $x \geq \ln(1+x) \geq x(1-x/2)$.*

Proof. The first inequality is due to $e^x \geq 1 + x$. For the next one, let $f(x) = \ln(1+x) - x(1-x/2)$, then $f'(x) = 1/(1+x) - (1-x)$. We have $f'(x) \geq 0$ since it is equivalent to $x^2 \geq 0$, so the minimum of $f(x)$ is $f(0) = 0$ or $f(x) \geq 0$. Therefore, $\ln(1+x) \geq x(1-x/2)$. □

Lemma 30. *For $n \in \mathbb{N}$ and $x \geq 0$, we have $1-(1-x)^n \geq 1 - e^{-xn} \geq \frac{xn}{1+xn}$.*

Proof. From $e^x \geq x + 1$, it follows that $1-(1-x)^n \geq 1 - e^{-nx} \geq 1 - (1+x)^{-n} = 1 - \frac{1}{1+\sum_{k=1}^{n}\binom{n}{k}x^k}$. the last term gives the first result. For $x \geq 0$, we have $x^k \geq 0$, so it suffices to cut off the sum at any $k \leq n$, the result is obtained with $k = 1$. Remark that cutting off at bigger values of k gives more precise bounds. □

Lemma 31. *For all $x \in \mathbb{R}$, $e^{2x} \geq (x+1)e^x > xe^x$ and for $x > 0$, $e^{2x}/x > e^x$.*

Proof. Multiplying e^x (> 0) to $e^x - x \geq 1 > 0$, we get $(e^x - x)e^x \geq e^x > 0$. Then adding xe^x to all sides gives the first result. The second one is obtained by dividing the first result with $x > 0$. □

Lemma 32 (Lemma 5 in [12]). *If $X \sim \mathrm{Bin}(\lambda, p)$ with $p \geq (i/\lambda)(1+\delta)$ and $i \geq 1$, then*

$$E\left[\ln\left(\frac{1+cX}{1+ci}\right)\right] \geq c\varepsilon$$

with $\varepsilon = \min\{1/2, \delta/2\}$ and $c = \varepsilon^4/24$.

Lemma 33 (Lemma 6 in [12]). *If $X \sim \mathrm{Bin}(\lambda, p)$ with $p \geq (i/\lambda)(1+\delta)$, then $E\left[e^{-\kappa X}\right] \leq e^{-\kappa i}$ for any $\kappa \in (0, \delta)$.*

Lemma 34 (Lemma 7 in [12]). *Let $Y_1, Y_2, \ldots, Y_n$ be n random variables where $Y_1 \leq Y_i$ for all $i \in [n]$. Then for any random variable $Z \in [n]$, it holds that $E[Y_Z] \geq E[Y_1]$.*