**UNIVERSITY OF ADELAIDE**

# Parameterised Complexity Analysis of Evolutionary Algorithms for Combinatorial Optimization Problems

*Author:*
Mojgan Pourhassan

*Principle Supervisor:*
Prof. Frank Neumann

*Co-Supervisor:*
Dr. Markus Wagner

A thesis submitted in fulfilment for the
degree of Doctor of Philosophy

in the
Optimisation and Logistics
School of Computer Science

May 2017

# Declaration of Authorship

I, Mojgan Pourhassan, certify that this work contains no material which has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, any partner institution responsible for the joint-award of this degree.

I give consent to this copy of my thesis when deposited in the University Library, being made available for loan and photocopying, subject to the provisions of the Copyright Act 1968.

I acknowledge that copyright of published works contained within this thesis resides with the copyright holder(s) of those works.

I also give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library Search and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

I acknowledge the support I have received for my research through the provision of an Australian Government Research Training Program Scholarship

Signed: _____

Date: _____


*"Dreaming when Dawn's Left Hand was in the Sky*
*I heard a voice within the Tavern cry,*
*'Awake, my Little ones, and fill the Cup*
*Before Life's Liquor in its Cup be dry.' "*

Omar Khayyam
Translated into English in 1859 by Edward FitzGerald

UNIVERSITY OF ADELAIDE

# *Abstract*

Engineering, Computer and Mathematical Sciences
School of Computer Science

Doctor of Philosophy

**Parameterised Complexity Analysis of Evolutionary Algorithms for Combinatorial Optimization Problems**

by Mojgan Pourhassan

Evolutionary algorithms are general problem solvers that have been successfully used in solving combinatorial optimization problems. However, due to the great amount of randomness in these algorithms, theoretical understanding of them is quite challenging. In this thesis we analyse the parameterized complexity of evolutionary algorithms on combinatorial optimization problems. Studying the parameterized complexity of these algorithms can help us understand how different parameters of problems influence the runtime behaviour of the algorithm and consequently lead us in finding better performing algorithms. We focus on two NP-hard combinatorial optimization problems; the generalized travelling salesman problem (GTSP) and the vertex cover problem (VCP). For solving the GTSP, two hierarchical approaches with different neighbourhood structures have been proposed in the literature. In this thesis, local search algorithms and simple evolutionary algorithms based on these approaches are investigated from a theoretical perspective and complementary abilities of the two approaches are pointed out by presenting instances where they mutually outperform each other. After investigating the runtime behaviour of the mentioned randomised algorithms on GTSP, we turn our attention to the VCP. Evolutionary multi-objective optimization for the classical vertex cover problem has been previously analysed in the context of parameterized complexity analysis. We extend the analysis to the weighted version of the problem. We also examine a dynamic version of the classical problem and analyse evolutionary algorithms with respect to their ability to maintain a 2-approximation. Inspired by the concept of duality, an edge-based evolutionary algorithm for solving the VCP has been introduced in the literature. Here we show that this edge-based EA is able to maintain a 2-approximation solution in the dynamic setting. Moreover, using the dual form of the problem, we extend the edge-based approach to the weighted vertex cover problem.

# *Acknowledgements*

I would like to express my sincere appreciation to my principle supervisor, Prof. Frank Neumann, for his motivation, patience, support, and priceless guidance during my PhD study. Without his insights and advices, this journey would have been extremely difficult for me, if not impossible.

My grateful thanks also go to my co-supervisor, Dr. Markus Wagner, for his encouragement, useful comments, and his positive attitude towards solving problems that I faced.

# Contents

# List of Figures

*To my family*

# Chapter 1

# Introduction

Bio-inspired computing techniques such as evolutionary algorithms [41] are general problem solvers that have been widely used in the last decades. Evolutionary algorithms (EAs) are inspired from natural evolution that makes creatures suit their environment during generations. The main idea of these algorithms is to (randomly) generate a number of solutions as the initial population, evolve some of them and replace the worst solutions of the population with new solutions that are better in terms of fitness. The variation operators that make the new solutions from old solutions are crossover and mutation and will be performed based on defined probabilities. This process will make the next generation and repeating the whole steps for many times can result in having a population with some solutions close to the optimum solution. These algorithms have been used very successfully during last decades for a wide range of applications such as combinatorial optimization problems.

Combinatorial optimization problems are the problems in which the solutions comprise a discrete set of variables and the goal is to find the optimal solution among the feasible set of solutions. The travelling salesman problem (TSP) and the vertex cover problem (VCP) are two well-known combinatorial optimization problems with several applications. Many combinatorial optimization problems such as TSP and VCP are NP-hard problems and need exponential time to be solved by a deterministic algorithm unless $P = NP$. Therefore, metaheuristics, which are problem-independent optimization techniques such as evolutionary algorithms, are usually used for solving instances of these problems with large inputs.

In spite of the vast practice of evolutionary algorithms and other bio-inspired strategies in different applications, the theoretical understanding of these algorithms is not advanced. These metaheuristics [49] are introduced to find acceptable solutions for difficult problems in reasonable time while they provide no guarantee for finding the

optimal solution. They are based on large amount of randomness, which to a great extent makes them able to search different parts of the search space. The great amount of randomness involved in these algorithms makes their theoretical analysis challenging. Despite the research that has taken place in this field in the last two decades, the working principle of these algorithms is not fully understood theoretically, and more progress is needed to clarify how they can be improved. Most of the theoretical research performed so far for bio-inspired algorithms, is on complexity analysis of evolutionary algorithms and ant colony optimization. Results have been achieved for classical polynomially solvable problems such as sorting, shortest path, minimum spanning trees and maximum matching as well as for some of the best known NP-hard combinatorial optimization problems such as vertex cover, makespan scheduling, and the travelling salesman problem. [6, 69, 77, 88, 91, 111, 112, 114].

Computational complexity analysis deals with the runtime behaviour of the algorithms with respect to the size of the problem instance. In some problems, the impact of the value of other parameters that are defined in different instances of the problem can also be investigated, in addition to the size of the input. This approach is called parametrized complexity analysis [36] and helps measuring the hardness of a problem instance in dependence of the studied additional structural parameters. In recent years, a parameterized complexity analysis has been carried on evolutionary algorithms solving several combinatorial optimization problems [18, 36, 76, 77, 88, 112]. For example, in [88], where the parameterized complexity analysis of the Euclidean TSP is studied, all the nodes are divided into two groups: Outer nodes (convex hull nodes) and inner nodes, and it is shown that when the number of inner nodes is not large, finding the optimal solution takes polynomial time with respect to the total number of nodes. Moreover, the authors of that paper have proven that under some reasonable geometric constraints, a simple EA can also solve a convex instance of the problem in polynomial time. In another work on parameterized complexity analysis [18], two evolutionary approaches for solving the generalized minimum spanning tree are investigated. In both approaches, the problem is split into two layers where the upper layer suggests a sub-solution by means of an evolutionary algorithm, and the lower layer completes the solution using a deterministic algorithm. In the presented analysis, it is shown that one of the approaches can find the optimal solution in polynomial time, if the number of clusters is small.

Theoretical understanding of metaheuristics such as evolutionary algorithms is important for several reasons. First, understanding the potentials and limitations of EAs in dealing with a problem helps us decide on selecting the proper solver for that problem. Moreover, rigorous analysis can help in adapting the EA parameters and designing

a better solver in a way that reduces the optimization time for our problem. Additionally, analysing different parameters of problem instances, can help us understand which algorithm is more suitable for a specific instance of the problem and lead to new identifications on the hardness of problem instances based on their parameters.

In this thesis, we conduct theoretical analyses of using local search and evolutionary algorithms for two combinatorial optimization problems, namely the generalized travelling salesman problem (GTSP) and the vertex cover problem (VCP). Hierarchical approaches (see Section 3.4 for a definition of hierarchical approaches) are quite popular for solving the GTSP. We investigate local search methods and provide parameterized analysis for simple evolutionary algorithms based on hierarchical approaches for this problem. In order to find a lower bound for one of the approaches, we present a Euclidean class of instances of the problem that cannot be solved in less than exponential time. Showing lower bounds for the Euclidean travelling salesman problem has been shown to be quite difficult. Englert et al. [42] have shown that there are instances of the Euclidean TSP for which finding a local optimal solution takes exponential time by means of a deterministic local search algorithm based on 2-opt. To our knowledge, an exponential lower bound for solving TSP by a stochastic search algorithm is available only for ant colony optimization in the non-Euclidean case [75].

After the analysis of GTSP, this thesis continues with investigating the behaviour of local search and evolutionary algorithms on the vertex cover problem. Theoretical analysis of evolutionary algorithms on this problem has been carried on in the last decade to an extent, with results for single-objective as well as multi-objective approaches (see Section 2.4 for a definition of multi-objective evolutionary algorithms). Friedrich et al. [48] have shown that the single-objective evolutionary algorithm (1+1) EA can not achieve a better than trivial approximation ratio in expected polynomial time. Furthermore, they have shown that a simple multi-objective evolutionary algorithm gives a factor $O(\log n)$ approximation for the wider classes of set cover problems in expected polynomial time. Further investigations regarding the approximation behaviour of evolutionary algorithms for the vertex cover problem have been carried out in [47, 95]. Kratsch and Neumann [77] have studied evolutionary algorithms and the vertex cover problem in the context of parameterized complexity. They have shown that a simple multi-objective evolutionary algorithm with a problem specific mutation operator is a fixed parameter evolutionary algorithm (see Section 4.6.1 for a definition of fixed parameter evolutionary algorithm) for this problem and finds 2-approximations in expected polynomial time. Jansen et al. [70] have shown that a 2-approximation can also be obtained by using an edge-based representation in a simple evolutionary algorithm

combined with a specific fitness function which is formulated based on matchings (Section 3.3.1 includes the definition of a matching and how to derive a 2-approximation vertex cover from a maximal matching).

In this thesis, the investigations on multi-objective evolutionary algorithms carried out in [77] are extended to the weighted vertex cover problem with integer weights. Furthermore, we investigate the edge-based representation of Jansen et al. [70] in connection with different fitness functions according to their approximation behaviour in the dynamic setting. The edge-based representation for this problem is inspired by the matching problem which is the dual problem of the vertex cover problem. The concept of duality is used in this thesis in a more general form to introduce an evolutionary algorithm that finds 2-approximations for the weighted vertex cover problem.

The outline of the thesis is as follows. We introduce evolutionary computation and local search methods in Chapter 2. A general definition of combinatorial optimization problems and the exact definition of the problems that are investigated in this thesis are presented in Chapter 3. The methods of analysing bio-inspired computing that are used in our theoretical analysis are presented in Chapter 4. Chapters 5 to 9 include the results of our theoretical analysis. Local search methods and evolutionary algorithms for GTSP are investigated in Chapters 5 and 6 respectively. The investigations on multi-objective evolutionary algorithms for the weighted vertex cover problem is presented in Chapter 7. We analyse the edge-based representation of Jansen et al. [70] with different fitness functions in the dynamic setting in Chapter 8. The concept of duality is used in an evolutionary algorithm for finding 2-approximations of the weighted vertex cover problem in Chapter 9. Finally, Chapter 10 presents the highlights of the study and concludes the thesis.

# Chapter 2

# Randomised Local Search and Evolutionary Computation

## 2.1 Introduction

Randomised local search and evolutionary computation are general problem solving methods that are widely used for NP-hard combinatorial optimization problems. The aim of using these methods is to provide good solutions for a problem that cannot be solved to optimality in polynomial time with existing algorithms. These methods are two of the broader class of *stochastic search algorithms* which search the decision space based on random selections.

As general problem solvers, these two approaches can be adapted to different problems, and this adaptation is easier when the search space of the problems are similar. We denote the search space (the decision space) of the problem by $S$. Randomised local search and evolutionary computation are iterative approaches that start with one (or more) solution(s) which are usually chosen at random, and come up with one (or more) new solution(s) in each iteration. The new solutions which are search points in $S$ need to be evaluated to guide the algorithm in searching good regions of the search space. The evaluation of a search point $x \in S$, is done by a fitness function $f(x) : S \rightarrow R$, with $R$ being the set of all possible values. This fitness function has to use the problem domain knowledge for giving sound and useful evaluations.

In this thesis, randomised local search and simple evolutionary algorithms are investigated from a theoretical perspective. This chapter, includes the definition of these approaches. We first describe local search in a general sense and introduce randomised local search in Sections 2.2 and 2.2.1 respectively. Then we give a brief description of

evolutionary algorithms, their components, and their subclasses in Section 2.3. We go into more details of a simple evolutionary algorithm called (1+1) EA, which is analysed for solving a couple of combinatorial optimization problems in this thesis. Moreover, we give an introduction to multi-objective evolutionary algorithms, together with a diversity mechanism that helps the algorithm search different parts of the search space.

## 2.2 Local Search

Local search is a widely used heuristic method for solving different kinds of optimization problems such as combinatorial optimization problems. A local search algorithm is an iterative algorithm that starts with an initial random solution, or a solution that is found by a heuristic. As the name suggests, the algorithm considers the local neighbourhood of the current solution $x$, and searches for a new solution $x'$ that is better than (or at least as good as) $x$ with respect to the fitness function. If such a solution is found in the neighbourhood of $x$, the algorithm replaces $x$ by $x'$, and the process of searching the neighbourhood of $x$ for a better neighbour starts over. The local neighbourhood of a search point needs to be predefined and should not be too large. The algorithm stops when it fails to find improvements in the neighbourhood of the current solution. At this point, either the optimal solution is found, or the algorithm is stuck to a local optimum.

The randomized local search is a local search in which at each iteration the new solution $x'$ is selected randomly among the solutions in the neighbourhood of $x$. This randomness makes the algorithm a stochastic search algorithm. The detailed description of the algorithm together with an example is presented in Section 2.2.1.

### 2.2.1 Randomized Local Search

Randomized local search (RLS) is one of the simplest stochastic search algorithms. Considering the current solution $x$, RLS chooses one solution $x'$ at each iteration in the neighbourhood of $x$ at random. The current solution $x$ is replaced by $x'$ at each iteration, if the fitness value of $x'$ is at least as good as the fitness value of $x$. Strict improvement can also be considered as the condition of accepting the new solution for this algorithm, where the fitness value of $x'$ needs to be better than the fitness value of $x$. The size of the neighbourhood that is defined to be used in RLS is very important. A neighbourhood that is too small results in fast convergence to a local optimum, and a neighbourhood that is too large makes it possible to choose a new solution that is very

**Algorithm 1:** RLS for a maximization problem with a bit-string representation

```
1 Choose x ∈ {0,1}^n uniformly at random;
2 while termination condition not satisfied do
3     x' = x;
4     Choose i ∈ {1, ··· , m} uniformly at random;
5     Flip x'_i;
6     if f(x') ≥ f(x) then
7         x = x';
```

different from the current solution; therefore, the algorithm can not guide the search properly.

We here define a simple combinatorial optimization problem named *OneMax*, and describe a RLS algorithm for solving it. The OneMax problem consists of $n$ binary variables where the goal is to maximize the number of ones among those variables. A solution is represented by a bit-string of size $n$, e. g. $x = (x_1, \cdots, x_n)$ where $x_i \in \{1, 0\}\ \forall i \in \{1, \cdots, n\}$, and the fitness function is the number of ones in that bit-string:

$$f(x) = \sum_{i=1}^{n} x_i$$

We consider the neighbourhood of Hamming distance 1 for a solution $x$ of this problem, which comprises all solutions that are different from $x$ in only one bit. Algorithm 1 presents a randomised local search on this problem, and in general on maximization problems with a bit-string representation. The algorithm starts with a random solution and iterates the main *while* loop until a desired condition is satisfied. This can be finding the optimal solution or an approximation of it, or exceeding the maximum number of fitness evaluations that the algorithm is allowed to perform. Lines 3-5 choose a new solution $x'$ from the defined neighbourhood of $x$ uniformly at random. Line 6 performs a fitness evaluation on the new solution to compare it with the current solution. Finally, the new solution replaces the current one if its fitness value is at least as good as that of the current solution.

## 2.3 Evolutionary Algorithm

Evolutionary Algorithms are stochastic search algorithms inspired by Darwin's principle of survival of the fittest in the nature. These general problem solvers work on a population of solutions or *individuals*, construct new solutions in an iterative manner,

and keep some of the fittest solutions and remove the others in each iteration depending on their survival policy. These algorithms have become quite popular since 1960s and have been used in real-word optimization applications.

Here we describe the main procedure of evolutionary algorithms as well as the terminology that is used for referring to important concepts of these methods. We use the OneMax problem, defined in Section 2.2.1, for presenting examples of each component that we explain.

The first step in using evolutionary algorithms for a problem is to define a *solution representation*. Depending on the problem and the variables that need to be adjusted in the optimization process, the representation can be a bit-string, or a string of integer or float variables or a combination of them. For the OneMax problem, similar to what we had in Section 2.2.1, an obvious representation is a bit-string of size $n$, such as $x = (x_1, \cdots, x_n)$, where $x_i$ corresponds to the value of the $i$th variable in the problem. Each solution $x$ needs to be evaluated somehow, so that the algorithm can compare different solutions. As explained briefly in Section 2.1, this evaluation is done by a function named the *fitness function*, denoted by $f(x) : S \to R$, where $S$ is the search space and $R$ is the set of all possible values of the fitness function. For OneMax, a simple and obvious fitness function can be $f(x) = \sum_{i=1}^{n} x_i$ which counts the number of ones in the bit-string. Here $S = \{0, 1\}^n$ and $R = \{0, 1, \cdots, n\}$. In practice, a fitness evaluation is usually costly; therefore, an optimization algorithm should find a good or close to optimum solution while minimizing the number of fitness evaluations.

In each iteration of the evolutionary process, some changes need to be done on the current solution(s) to produce new solution(s) (*offspring*). These changes are performed by *variation operators* which are mainly *crossover* and *mutation* in an evolutionary algorithm, and should be adjusted to the solution representation. Crossover is done on two selected parents and produces one or more offspring, while mutation is performed on one parent and produces one child. For example, in the case of OneMax, one crossover operator that can be considered is the single-point crossover, in which one position in the two parents is selected and the parent bit-strings are split into two substrings from that position. The substrings are then combined to produce two new solutions as children. If $x = (x_1, \cdots, x_n)$ and $y = (y_1, \cdots, y_n)$ are the two parents and the selected position in the bit-string is $p$, then the two children can be $x' = (x_1, \cdots, x_p, y_{p+1}, \cdots, y_n)$ and $y' = (y_1, \cdots, y_p, x_{p+1}, \cdots, x_n)$.

A simple mutation operator for the bit-string representation could also be defined as flipping each bit of the parent with probability $\frac{1}{n}$. This mutation operator flips one bit at each step on average, but can also make several changes on the given solution with a

smaller probability at each step. This is one of the main differences between evolutionary algorithms and local search methods, i. e. in a local search algorithm only solutions in the predefined local neighbourhood of the current solution can be considered, while the mutation operator in an evolutionary algorithm makes it possible to jump to farther search points in the search space. In case of a discrete search space, it is possible to move from any search point to any other search points with a positive probability. This property makes evolutionary algorithms less likely to stick to a solution that is a local optimum for a randomised local search algorithm.

The other main difference between local search and evolutionary algorithms is that in local search the algorithm keeps only one solution at each iteration and produces only one new solution, while in an evolutionary algorithm usually a *population* of more than one solution is kept and more than one solutions are produced as offspring. This allows having a diverse set of solutions, i. e. different solutions from different parts of the search space. A $(\mu + \lambda)$EA denotes an evolutionary algorithm in which the population size is $\mu$ and the offspring size is $\lambda$. Although evolutionary algorithms usually have $\mu > 1$ and $\lambda > 1$ in practice, it is also possible to work with $\mu = 1$ and $\lambda = 1$ which results in (1+1) EA, defined in Section 2.3.1.

After the offspring are produced, in order to choose $\mu$ solutions from the current $\mu + \lambda$ solutions as the next generation, and also in order to choose some solutions among the current population as parents, evolutionary algorithms need *selection methods*. One of the most important selection methods is the *fitness-proportional selection method*. Assuming that the fitness function needs to be maximized and all values it returns are positive, in this method the probability of choosing a solution $x$ among $k$ solutions $x^j, j \in \{1, \cdots, k\}$ is

$$\frac{f(x)}{\sum_{j=1}^{k} f(x^j)}.$$

This results in selecting fitter solutions with a higher probability compared to the solutions that have a small value of fitness.

The other selection method that is widely used is *tournament selection* in which a number of solutions are chosen uniformly at random to comprise each tournament, and the fittest solution in each tournament is selected for parent selection or next generation. The number of tournaments that are made is equal to the number of individuals that are selected by this method.

Depending on the problem and the search space, there are different types of evolutionary algorithms that differ in some implementation details such as the representation that is used. For example, in the case of problems with continuous (real-valued)

**Algorithm 2:** (1+1) EA for a maximization problem with a bit-string representation

```
1 Choose x ∈ {0,1}^n uniformly at random;
2 while termination condition not satisfied do
3     x' = x;
4     ∀i ∈ {1, ··· , m} flip x'_i with probability 1/n;
5     if f(x') ≥ f(x) then
6         x = x';
```

search space, *evolutionary strategies (ES)* are used for optimization; whereas, for a discrete search space where bit-strings or strings of integer values can be used as the representation, genetic algorithms (GA) are used. Another type of evolutionary algorithm that has gained some attention in recent decades is genetic programming (GP) in which computer programs (or graphs in general) for given tasks are tried to be constructed as solutions. For a comprehensive description of different types of evolutionary algorithms, refer to the text book of Eiben and Smith [41]. In this thesis, we only analyse simple evolutionary algorithms on discrete search spaces. The rest of this section includes the description and the pseudo-code of the (1+1) EA for a maximization problem as a simple evolutionary algorithm. Moreover, multi-objective evolutionary algorithms are defined together with a method that helps them with the diversity of the population.

### 2.3.1 (1+1) EA

(1+1) EA is possibly the simplest evolutionary algorithm that can be defined. In this algorithm, the population consists of one solution, and only one solution is generated as the offspring at each iteration. The current solution $x$ is replaced by the new solution $x'$, if $x'$ is better than or at least as good as $x$ with respect to the fitness function. The only difference between this algorithm and the RLS of Section 2.2.1 lies in constructing the new solution. In RLS a solution in the defined neighbourhood of $x$ is selected, while in (1+1) EA each part of the representation (e. g. each bit of the bit-string in the OneMax example) is mutated with a small probability. A pseudo-code of (1+1) EA for the OneMax problem is presented in Algorithm 2. In this algorithm, constructing the new solution is done in lines 3-4.

## 2.4 Multi-Objective EA

Evolutionary algorithms are quite popular for multi-objective optimization where the quality of solutions is assessed using more than one objective. If we denote the number of objectives by $k$, in these problems we deal with an objective space of $k$ dimensions and the fitness function returns an objective vector of $k$ values, i.e. $f(x) = (f_1(x), \cdots, f_k(x))$. In a maximization problem, we say that a solution $x$ (weakly) dominates another solution $y$ and denote it by $f(x) \succeq f(y)$, if $f(x) \neq f(y)$ and $f_i(x) \geq f_i(y), \forall i \in \{1, \cdots n\}$. A solution $x$ that is not dominated by any other solutions is called a *Pareto optimal search point*.

In a multi-objective optimization problem, we are interested in the set of Pareto optimal search points and their objective vectors. The set of Pareto optimal search points is called the *Pareto set*, whereas the set of objective vectors of them is called the *Pareto front*. Note that each point in the objective space can be the objective vector of more than one search point in the search space. The goal of the optimization algorithm is to find the Pareto front or a set of objective vectors that are close to it, together with at least one search point from the Pareto set for each objective vector. Since it is desirable to obtain a range of trade-offs between objectives, the diversity of the found objective vectors is an important point in the optimization process.

As a simple example, we describe the multi-objective OneMax-ZeroMax problem which is defined on a bit-string of size $n$, and the goal is to maximize the number of ones and the number of zeros in the bit-string. Note that the objectives are conflicting in this problem: $f_1(x)$ denotes the number of ones, and $f_2(x)$ denotes the number of zeros. The Pareto front in this problem consists of the set $\{(i, n-i) \mid 0 \leq i \leq n\}$, and any search point with $i$ ones and $n-i$ zeros corresponds to the objective vector $(i, n-i)$ for $0 \leq i \leq n$.

Several multi-objective evolutionary algorithms have been introduced in the literature, starting with vector-evaluated genetic algorithm (VEGA), which was presented in 1984 by Schaffer [106]. In this algorithm, the survival selection is as follows. For each of the $k$ objectives that need to be optimised, a subset of the best existing solutions with respect to that objective is selected by proportional selection. Then these subsets are combined to make the new population. With this approach, the diversity is preserved to a good extend by including good solutions for each objective. However, solutions that are moderately good for all objectives and can be very useful do not survive, if they are not very good for any single objective.

Other multi-objective evolutionary algorithms were introduced later which guide the search towards finding non-dominated solutions explicitly. These algorithms include

Fonseca and Fleming's multi-objective genetic algorithm (MOGA) [46] which considers the number of solutions in the population that are dominated by an individual plus one, as the individual's fitness value. Non-dominated sorting genetic algorithm (NSGA) [107] and its improved version, NSGA-II [25] are also two other well-known multi-objective evolutionary algorithms that focus on the diversity as well as finding near optimal solutions. In NSGA, the population is divided into subsets as the following. The first subset is the Pareto front solutions of the whole population. Eliminating these solutions from the population, the next Pareto front solutions is found from the population which form the second subset and so on. For each solution, the algorithm assigns the number of solutions in inferior subsets, as the fitness. By performing a selection method that is likely to choose solutions with higher fitness, solutions of subsets with lower numbers are selected, which directs the search towards the Pareto optimal. Furthermore, using a parameter, $\sigma_{share}$, that indicates the neighbourhood size, a *fitness sharing strategy* is applied to change the fitness of solutions based on how close they are positioned in the objective space. This strategy helps in maintaining the diversity by decreasing the fitness of some individuals that are too close to each other.

One issue in MOGA and NSGA is that good solutions can be lost in their survivor selection procedure. In order to fix this, NSGA-II [24] was proposed, in which solutions of subsets (or fronts) with lower numbers are selected in an elitist approach, to perform a $(\mu + \lambda)$ survivor selection. If some solutions need to be selected and others need to be eliminated from the last considered subset, then the distance of solutions from their nearest neighbours in the same subset is considered as a metric, to help with the selection. The aim of using this new metric is to maintain the diversity when some solutions are positioned very close to each other. This method does not need specifying the sharing parameter that was used in NSGA. It is worth to mention that NSGA-II also outperforms NSGA with respect to computational complexity.

Global SEMO is another multi-objective evolutionary algorithm, which we describe in more detail in the rest of this section. This algorithm is later analysed for the vertex cover problem in Chapter 7.

### 2.4.1 Global SEMO

Here we describe a simple multi-objective evolutionary algorithm called Global SEMO which is presented in Algorithm 3 for a multi-objective maximization problem. In this algorithm we have a population of solutions, $P$, which is initialized with one randomly selected solution. At each iteration, one solution is randomly selected from this population, the variation operator is performed on it (line 6 describes the mutation operator),

**Algorithm 3:** Global SEMO for a multi-objective maximization problem

1 Choose $x \in \{0,1\}^n$ uniformly at random;
2 $P \leftarrow \{x\}$;
3 **while** *termination condition not satisfied* **do**
4  Choose $x \in P$ uniformly at random;
5  $x' = x$;
6  $\forall i \in \{1, \cdots, m\}$ flip $x'_i$ with probability $\frac{1}{n}$;
7  **if** $\nexists y \in P \mid f(y) \succeq f(x')$ **then**
8  $P \leftarrow \{x'\}$;
9  delete all other solutions $z \in P$ where $f(x') \succeq f(z)$ from $P$;

and the survival selection procedure is done afterwards (lines 7-9). In the survival selection procedure, the new solution, $x'$, is checked whether it is dominated by at least one other solution in the population. If $x'$ is not dominated by any other solutions in $P$, then it is added to $P$, and all other solutions that are dominated by $x'$ are removed from $P$. In other words, the algorithm keeps the non-dominated solutions that are discovered so far in $P$.

In the problem of OneMax-ZeroMax, the size of the population grows to the size of Pareto front, which is at most $n+1$ and is linear with respect to the input size. If the size of the Pareto front is too large (e. g. exponential), usually a diversity mechanism is used to keep the population size within a polynomial range, while making it diverse enough to cover most parts of the Pareto Front. One such mechanism is described in the following section.

### 2.4.2 $\epsilon$-Dominance: A Diversity Mechanism

In multi-objective optimization problems where the size of the Pareto front is exponential with respect to the input size, an optimization algorithm would better find a set of solutions that has a polynomially bounded size, and is a good approximation of the Pareto Front. In this situation, evolutionary algorithms need a diversity mechanism to help their limited population cover most parts of the Pareto front.

One approach for dealing with this problem is using the concept of $\varepsilon-$dominance [78]. The concept of $\varepsilon-$dominance has previously been proved to be useful for coping with exponentially large Pareto fronts in combinatorial optimization problems [64, 90]. Having a maximization problem and two search points $x$ and $y$ with objective vectors $f(x) = (f_1(x), \cdots, f_m(x))$ and $f(y) = (f_1(y), \cdots, f_m(y))$, $f(x)$ $\varepsilon-$dominates $f(y)$, denoted by $f(x) \succeq_\varepsilon f(y)$, if for all $i \in \{1, \cdots, m\}$ we have $(1+\varepsilon)f_i(x) \leq f_i(y)$.

The goal of this approach is to find an approximation of the Pareto front. Here, recalling that $S$ is the search space, the $\varepsilon-$*Pareto front*, denoted by $PF^*_\varepsilon$, is defined as a subset of the Pareto front where

$$\forall x \in S,\ \exists v \in PF^*_\varepsilon \mid v \succeq_\varepsilon f(x).$$

Moreover, the corresponding Pareto set of $PF^*_\varepsilon$ is called the $\varepsilon-$*Pareto set*. Note that there can exist several $\varepsilon-$Pareto fronts and $\varepsilon-$Pareto sets.

In order to find an approximation of the Pareto front in this approach, the objective space is partitioned into a polynomial number of boxes in which all solutions $\varepsilon-$dominate each other, and at most one solution from each box is kept in the population. The partitioning of the objective space is done by placing a hyper-grid in that space such that the number of partitions is logarithmic with respect to the maximum value that each objective can take. In this approach, the population size is polynomially bounded, and the solutions that are kept in the population comprise an $\varepsilon-$Pareto set of the search points that the algorithm has found so far. For details of the partitioning and the proof of the last two statements refer to [78].

The boxing of [64] which we describe here, presents such a partitioning. There, each objective vector is mapped to the index of the box that it is placed in. Assuming that the objective space is positive and normalised, i.e. $\forall i \in \{1, \cdots, m\},\ f_i(x) > 1$, the mapping is done as:

$$b(x) = (b_1(x), \cdots, b_m(x))\ ,\ b_i(x) = \left\lfloor \frac{\log(f_i(x))}{\log(1+\varepsilon)} \right\rfloor$$

and $b(x)$ is called the *box index vector* of the search point $x$.

In the evolutionary algorithm that uses the concept of $\varepsilon-$dominance in [64], a non-dominated offspring that its box index vector is not dominated by any other existing box index vectors is accepted, and at each iteration, all solutions that their box index vector is dominated are deleted from the population. Inspired by this partitioning and algorithm, a similar boxing and multi-objective evolutionary algorithm is defined and analysed for the vertex cover problem is Chapter 7.

## 2.5 Conclusion

In this chapter, we presented an introduction to two of the well-studied general problem solver methods, namely local search and evolutionary algorithms. We went into more details of the randomised local search and (1+1) EA, which is possibly the simplest existing evolutionary algorithm. Comparing the randomness that is used in these

two algorithms when making new solutions, RLS is more likely to get stuck in local optimums than (1+1) EA.

After describing evolutionary algorithms and presenting a simple algorithm from this class of algorithms, we turned our attention to multi-objectiveness, and described Global SEMO as a multi-objective evolutionary algorithm. Moreover, we presented $\varepsilon-$dominance, a diversity mechanism that helps the algorithm find solutions from different parts of the search space.

# Chapter 3

# Combinatorial Optimization Problems

## 3.1 Introduction

A combinatorial optimization problem consists of finding the best feasible solution (with either minimum or maximum value of an objective function) when the solution space of the problem is discrete and the feasibility is determined by satisfying some given constraints. For any instance of a problem a specified parameter setting is given. The formal definition of a combinatorial optimization problem is as follows. Given a triple $(S, f, \Omega)$, where $S$ is the search space, $f$ is the objective function, and $\Omega$ is the set of constraints, the goal is to find a globally optimal solution that fulfils all constraints [91].

The optimization time of an algorithm, i. e. the time that the algorithm needs to find the optimal solution, is analysed with respect to the input size. The input of the problem, which is often a graph or a set of integers for a combinatorial optimization problem, has to be represented as a sequence of symbols of a finite alphabet. The input size is the length of this sequence, for any instance of the problem. The search space of a combinatorial optimization problem is most of the times exponential with respect to the size of the problem, and for many of these problems, a polynomial-time algorithm that finds the optimal solution can not be found unless $P = NP$. As a result, finding good approximations of the optimal solutions are of great importance, and approximation algorithms are widely considered for these problems.

The obtained solutions of approximation algorithms should be close to the optimal solution in terms of cost. In order to clearly specify how close they should be, we use the term approximation ratio, which is defined in [16] as follows.

**Definition 3.1** (**Approximation Ratio** [16])**.** An algorithm for a problem has an *approximation ratio* of $\rho(n)$ if for any input of size $n$, the cost $C$ of the solution produced by the algorithm is within a factor of $\rho(n)$ of the cost $C^*$ of an optimal solution:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n)$$

Knapsack problem, vertex cover problem, travelling salesman problem and makespan scheduling are some well-known combinatorial optimization problems for which exact and approximation algorithms have been introduced [115]. In this chapter we describe two of the well studied combinatorial optimization problems, the travelling salesman problem and the vertex cover problem. Then we define the concept of hierarchical approaches for solving combinatorial optimization problems, and describe the generalized travelling salesman problem as an example of problems that hierarchical approaches are popular for solving them.

## 3.2 The Travelling Salesman Problem

The travelling salesman problem (TSP) is a well-known NP-hard combinatorial optimization problem. The problem is to find a Hamiltonian cycle of minimum cost in a complete graph. The formal definition of the problem follows.

**Definition 3.2** (**The travelling salesman problem (TSP) [115]**)**.** Given a complete graph with non-negative edge costs, find a minimum cost cycle visiting every vertex exactly once.

TSP has many applications in logistics, scheduling and manufacturing such as the vehicle routing [4] and the computer wiring [68]. There are some exact algorithms developed for this well-studied problem among which one of the most important ones is the *cutting plane method*, introduced by Dantzig et al. [22] in 1954.

This method suggests a relaxation on the main problem and uses integer-linear programming to solve the relaxed problem. The solutions for this relaxed problem may or may not denote a tour on the given graph, i. e. it may be in the feasible space of the TSP problem or not. If the optimal solution of the relaxed problem is a tour, then it is also an optimal solution for the original TSP problem. If it is not a tour, then it gives a lower bound on the cost of the optimal solution of the TSP problem, and can be eliminated from the feasible space of the relaxed problem by an additional linear constraint or a *cut*. Dantzig et al. [22] add such constraints, which they call sub-tour elimination

constraints, in an iterative fashion and solve the LP problem again, until a tour with no structural problems is found. The heart of the TSP solver 'Concorde' [3] is an efficient implementation of this method; although, other successful techniques and heuristics are also used in different modules of this powerful tool. For details of the techniques and algorithms that are used in Concorde, refer to [5].

During 1960s, a different exact method was emerged for solving TSP that Little et al. [82] named the *branch-and-bound method*. Similar to the cutting plane method, in the branch-and-bound method the optimization problem is relaxed. Moreover, the problem is split into some (possibly more than two) sub-problems with additional constraints, branching the feasible space of the relaxed problem. Improvements of this method were made by Held and Karp [60, 61] in 1970, beating the results of [22]. Moreover, an integration of the cutting plane method into the branch and bound framework has been introduced by Hong [63] and used in [84]. This method is named the *branch and cut method* in [96] which was the first computational success of this approach [5].

Another important exact approach for TSP is the *dynamic programming approach* studied in [10, 11, 50, 59]. Held and Karp [59] have shown that an n-city TSP can be solved in time proportional to $n^2 2^n$.

Assuming that $\alpha(n)$ is a polynomial function of $n$, it is proved that the general TSP can not be approximated within a factor of $\alpha(n)$ in polynomial time, unless $P = NP$ [115]. Nevertheless, there are some approximation algorithms for Metric TSP in which triangle inequality is satisfied. One such algorithm [14] finds an approximation of ratio $\frac{3}{2}$ in time $O(n^4)$.

In addition to exact and approximate algorithms for solving TSP, a lot of heuristic methods were also introduced for this problem. One of the most successful ones is the Lin-Kernighan heuristic [81], built based on the classical 2-opt operator for the TSP, which replaces 2 edges of a given tour. This heuristic considers a sequence of a variable number of 2-opt moves, performed one after another. In other words, a variable number of changes are performed on the tour in each iteration of the heuristic for finding improvements.

The other approach for solving TSP is using metaheuristics such as evolutionary algorithms and ant colony optimization. Theoretical results on a number of these algorithms for the TSP have been presented in [42, 75, 87, 113]. Despite this progress and the advancement of methods for analysing these algorithms, understanding and analysing the run of even simple metaheuristics on classical problems such as run of local search

on TSP remains challenging, and obtaining theoretical results that match practical experience is to a large extend an open problem. For more details on theoretical analysis of these algorithms when solving TSP, refer to Nallaperuma's PhD thesis [87].

## 3.3 The Vertex Cover Problem

The vertex cover problem is another well-known NP-hard combinatorial optimization problem with various applications in scheduling, networking, bioinformatics, computational biology and detection of race conditions [1, 8, 51, 92, 97]. The input of the problem is a graph and the goal is to select a minimum subset of nodes, $V_C$, that covers all edges, i.e. every edge has at least one endpoint incident at $V_C$. This problem is defined on undirected graphs in which each edge $e$ is denoted by a set of two vertices, i.e. $e = \{u, v\}$ where $u$ and $v$ belong to the set of nodes.

The cardinality (or classical) vertex cover problem can be formally defined as:

**Definition 3.3** (**The Cardinality Vertex Cover Problem** [115])**.** Given an undirected graph $G = (V, E)$ with vertex set $V = \{v_1, \ldots, v_n\}$ and edge set $E = \{e_1, \ldots, e_m\}$, find the minimum cardinality vertex cover, i.e. a subset of vertices, $V_C \subseteq V$, such that $\forall e \in E, e \cap V_C \neq \emptyset$.

In the general form of the problem or the weighted vertex cover problem, defined below, weights or costs are assigned to the vertices and the goal is to find a cover of minimum weight:

**Definition 3.4** (**The Vertex Cover Problem** [115])**.** Given an undirected graph $G = (V, E)$ with vertex set $V = \{v_1, \ldots, v_n\}$ and edge set $E = \{e_1, \ldots, e_m\}$, and a positive weight function on vertices $w : V \to \mathbb{Q}^+$, the goal is to find the minimum cost vertex cover, i.e. a subset of vertices, $V_C \subseteq V$, such that $\forall e \in E, e \cap V_C \neq \emptyset$.

Approximation algorithms have been presented for the vertex cover problem. One simple 2-approximation algorithm for the cardinality vertex cover problem is based on finding a maximal matching. This approach is presented in Section 3.3.1. In a more general approach, the linear programming formulation and the dual problem of the vertex cover problem are considered, and a 2-approximation is found based on a maximal dual solution. We present this approach in Section 3.3.2.

**Algorithm 4:** Cardinality Vertex Cover 2-Approximation Algorithm

```
1  E_M ← ∅;
2  while E ≠ ∅ do
3  |   Pick an edge e = {p, q} ∈ E;
4  |   E_M ← E_M ∪ {e};
5  |   for ∀e' ∈ E where e' ∩ e ≠ ∅ do
6  |   |   E ← E \ e';
7  C ← ∅;
8  for ∀e = {p, q} ∈ E_M do
9  |   C ← C ∪ {p, q}.
10 return C
```

### 3.3.1 A 2-approximation algorithm for the cardinality vertex cover problem

One approximation algorithm with an approximation ratio of two for the cardinality vertex cover problem, represented in Algorithm 4, is based on finding a maximal matching. Given a graph $G = (V, E)$, a *matching* is a subset of edges, $M \subset E$ so that no two edges of $M$ share an endpoint [115]. A matching that is maximal under inclusion is called a *maximal matching*, and can be computed in polynomial time by a greedy algorithm that picks a random edge at a time and removes its endpoints from the graph, until there are no edges left. Here we show that the output of Algorithm 4 which is the set of matched vertices of a maximal matching, form a vertex cover with approximation ratio of 2. First we prove that the set of vertices, $C$, returned by the algorithm is a vertex cover. Then we show that the approximation ratio of this solution is 2.

The while loop in the algorithm finds the maximal matching $E_M$. Since this matching is maximal, all other edges of the graph have at least one common endpoint with one or two of the edges in $E_M$; otherwise, they could be added to the matching. In other words, the set of vertices included in the matching, $C$, form a vertex cover.

Now we show that the vertex cover $C$, is a 2-approximate solution. Observe that $C$ includes two vertices for each edge of $E_M$. Let $|E_M| = k$, which implies $|C| = 2k$. Since $E_M$ is a matching, the including edges do not have shared nodes; therefore, in order to cover all edges of $E_M$, $k$ vertices are required. This means that an optimal solution that covers all edges of the given graph needs at least $k$ vertices, implying that the approximation ratio of solution $C$ is at most 2; since $|C| = 2k$.

### 3.3.2 Linear programming formulation of the vertex cover problem and its dual problem

Consider an optimization problem whose requirements can be represented by linear relationships. These problems are referred to as *linear programming (LP)* problems, and the standard formulation of them with a goal of minimizing the objective function is represented as

$$\begin{aligned} & \min \ \sum_{i=1}^{n} c_i x_i \\ s.t. \quad & \sum_{i=1}^{n} a_{ji} x_i \geq b_j \quad j = 1, \cdots, m \\ & x_i \geq 0, \quad i = 1, \cdots, n \end{aligned}$$

where $c_i$, $a_{ji}$ and $b_j$ are given rational numbers.

Any LP problem (which we refer to as the *primal* problem) has a *dual* form, which is also an LP problem and helps with finding bounds on the objective values of the primal problem. When the primal problem is a minimization problem, the dual problem considers a positive linear combination of the constraints, and looks for lower bounds on the minimum possible value of the primal objective function. In the above formulation of the primal problem, there are $m$ constraints of this form:

$$a_{j1}x_1 + a_{j2}x_2 + \cdots + a_{jn}x_n \geq b_j \qquad j = 1, \cdots, m$$

If we multiply both sides of each of these $m$ constraints by a coefficient $y_j$, we have:

$$a_{j1}y_j x_1 + a_{j2}y_j x_2 + \cdots + a_{jn}y_j x_n \geq b_j y_j \qquad j = 1, \cdots, m$$

Consider the sum of all these constrains:

$$\left(\sum_{j=1}^{m} a_{j1}y_j\right) x_1 + \left(\sum_{j=1}^{m} a_{j2}y_j\right) x_2 + \cdots + \left(\sum_{j=1}^{m} a_{jn}y_j\right) x_n \geq \sum_{j=1}^{m} b_j y_j$$

If we manage to set the values of $y_j$ so that the left side of this inequality is less than the objective function, then the right side of the inequality gives a lower bound on the minimum value that the primal objective function can take. Finding the greatest lower bound under this condition is the definition of the dual problem, which is formulated as:

$$\begin{aligned} & \max \sum_{j=1}^{m} b_j y_j \\ s.t. \quad & \sum_{j=1}^{m} a_{ji} y_j \leq c_i \quad i = 1, \cdots, n \\ & y_j \geq 0, \quad j = 1, \cdots, m \end{aligned}$$

In contrast to what we have for a primal minimization problem, when the primal problem is a maximization problem, the dual problem finds upper bounds of the value of the primal objective function. For more explanations on primal and dual forms of an LP problem, and how to derive the dual form from the primal form refer to [115]. Considering these formulations, *the Weak Duality Theorem* described below, helps with finding lower bounds of any feasible solution of the primal problem. The reader can find the proof of this theorem in [115].

**Theorem 3.5** (The Weak Duality Theorem [115]). *If $x = (x_1, \cdots, x_n)$ and $y = (y_1, \cdots, y_m)$ are feasible solutions for the primal and dual problem respectively, then*

$$\sum_{i=1}^{n} c_i x_i \geq \sum_{j=1}^{m} b_j y_j.$$

When some or all of the variables in an LP problem are restricted to be integers, the problem is referred to as an *integer linear programming (ILP)* problem. Here we define the ILP and LP formulation of the vertex cover problem which help us in finding 2-approximations of the optimal solution. The 2-approximation algorithm for the classical vertex cover problem in which a maximal matching is found (Section 3.3.1), can be considered as a special case of the approach that we describe in this section.

The common representation that is used for solutions of the vertex cover problem is the standard node-based representation, in which a solution $x = (x_1, \ldots, x_n)$ is a bitstring of size $n$, where $x_i = 1$ iff the node $v_i$ is chosen. With this representation, the ILP formulation for this problem is:

$$\begin{aligned} & \min \sum_{i=1}^{n} w(v_i) \cdot x_i \\ s.t. \quad & x_i + x_j \geq 1 \quad \forall (i,j) \in E \\ & x_i \in \{0,1\} \quad \forall i \in \{1, \cdots, n\} \end{aligned}$$

In the linear programming relaxation of the problem, *the fractional weighted vertex cover problem*, the constraint $x_i \in \{0, 1\}$ is relaxed to $x_i \in [0, 1]$. We denote the cost of the optimal solution for the original problem and the relaxed version of the problem by $OPT$ and $OPT^*$ respectively. Observe that $OPT^* \leq OPT$.

Using the concept of duality and the Weak Duality Theorem, 2-approximations of the vertex cover problem can be obtained. Here we explain the situation where the weights in the vertex cover problem are restricted to be integer values. The dual of the relaxed covering problem in which the weights can only take integer values, is a packing problem formulated as the following, where $s_j \in \mathbb{N}^+$ denotes a weight on the edge $e_j$:

$$\begin{aligned} & \max \sum_{j=1}^{m} s_j \\ s.t. \quad & \sum_{j \in \{1,\cdots,m\} | e_j \cap \{v\} \neq \emptyset} s_j \leq w(v) \quad \forall v \in V \end{aligned}$$

In other words, the dual problem is to maximize the sum of weights on all edges, provided that for each vertex, the sum of weights of edges incident to that vertex is at most equal to the weight of that vertex.

Let $s = (s_1, \cdots, s_m)$, be a maximal feasible solution for the dual problem with a cost of $Cost_D$. Since $s$ is a maximal solution, none of the edges can be assigned a greater weight without violating a constraint. Therefore, for at least one vertex of each edge, $v$, we have

$$w(v) = \sum_{j \in \{1,\cdots,m\} | e_j \cap \{v\} \neq \emptyset} s_j$$

As a result, the set of nodes for which the above equality holds, $C = \{v \in V \mid w(v) = \sum_{j \in \{1,\cdots,m\} \mid e_j \cap \{v\} \neq \emptyset} s_j\}$, is a vertex cover. The cost of this vertex cover, $Cost_P$, is at most twice the weight of all edges in the dual solution. Therefore, $Cost_P \leq 2 \cdot Cost_D$. Moreover, since $s$ is a feasible solution, according to Weak Duality Theorem (Theorem 3.5), $Cost_D \leq OPT$, which results in $Cost_P \leq 2 \cdot OPT$, i.e. set $C$ is a 2-approximation for the weighted vertex cover problem.

Constructing maximal solutions for the dual problem has been used in a number of algorithms for finding 2-approximations of the weighted vertex cover problem, e.g. Bar-Yehuda and Evan's greedy algorithm [9] and Clarkson's greedy algorithm [15]. Bar-Yehuda and Evan's greedy algorithm is presented in Algorithm 15 in which C represents the vertex cover that the algorithm finds. A formal proof of the approximation ratio of the solution obtained by this approach can be found in Theorem 8.4 of [39] (represented in Theorem 3.6 below). There, the output of a specific algorithm is studied as

**Algorithm 5:** Bar-Yehuda and Evan's greedy algorithm

**forall the** $v \in V$ **do** $W(v) \leftarrow w(v)$;
**forall the** $j \in \{1, \cdots, m\}$ **do** $s_j \leftarrow 0$;
$C \leftarrow \emptyset$;
**while** $E \neq \emptyset$ **do**
    Pick an edge $e_j = (p, q) \in E$;
    Suppose $W(p) \leq W(q)$;
    $s_j \leftarrow W(p)$;
    $W(q) \leftarrow W(q) - W(p)$;
    **for** $\forall e = (p, v) \in E$ **do**
        $E \leftarrow E \setminus e$;
    **end**
    $C \leftarrow C \cup \{p\}$.
**end**
**return** $C$

the maximal dual solution, but the presented proof is valid for Theorem 3.6 with any given maximal solution $s$.

**Theorem 3.6.** *Consider $s$, a maximal feasible solution for the dual problem of the relaxed weighted vertex cover problem. The vertex set*

$$C = \{v \in V \mid w(v) = \sum_{j \in \{1, \cdots, m\} \mid e_j \cap \{v\} \neq \emptyset} s_j\}$$

*is a 2-approximation for the original weighted vertex cover problem.*

## 3.4 Hierarchical Approaches for Solving Combinatorial Optimization Problems

One important strategy in constructing efficient algorithms for complex problems is to split the problem into some (usually two) layers and solve each layer by considering the sub-solution that its upper layer is suggesting. The upper layer can fix a sub-solution independent of the lower layer, but the sub-solutions of all layers affect the cost of the eventual complete solution. This approach, which is called hierarchical approach, is studied in different single-objective [72, 79] as well as multi-objective problems [26, 27].

The generalised minimum spanning tree (GMST) and the generalised travelling salesman problem (GTSP) are two of the combinatorial optimization problems that hierarchical approaches are quite popular for solving them. Hu and Raidl [65] have presented two hierarchical approaches for GMST. In both approaches, the upper layer is solved by

means of an EA and the lower layer uses a deterministic algorithm. The parameterized complexity analysis of these approaches is carried out in [18] which is the first paper on runtime analysis of evolutionary algorithms for hierarchical optimization problems. In the presented analysis, upper and lower bounds are found for both algorithms and it is shown that one of the approaches is a fixed-parameter evolutionary algorithm (Section 4.6.1) with respect to the number of clusters. Hu and Raidl [66] have also presented two hierarchical approaches for GTSP by which we investigate in this thesis. Our goal is to analyse these hierarchical approaches theoretically to give us a better understanding of the behaviour of them when solving the GTSP. We have conducted a parameterized computational complexity analysis on these two approaches with simple evolutionary algorithms. The formal definition of the problem and an illustration of the two hierarchical approaches for this problem follow.

### 3.4.1 The Generalised Travelling Salesman Problem

The generalised travelling salesman problem is an example of combinatorial optimization problems, for which hierarchical approaches are presented to solve the problem. This problem has applications in several fields such as planning and routing. Definition 3.7 gives a formal definition of this problem.

**Definition 3.7.** Given a complete undirected graph $G = (V, E, c)$ with cost function $c\colon E \to \mathbb{R}^+$ on the edges, and a partitioning of the set of nodes $V$ into $m$ clusters $V_i$, $1 \leq i \leq m$ such that $V = \bigcup_{i=1}^{m} V_i$ and $V_i \cap V_j = \emptyset$ for $i \neq j$. The aim is to find a tour of minimum cost that contains exactly one node from each cluster.

A candidate solution for this problem consists of two parts. The set of *spanning nodes*, $P = \{p_1, \ldots, p_m\}$ where $p_i \in V_i$, and the *permutation* of the nodes, $\pi = (\pi_1, \ldots, \pi_m)$, which is also a Hamiltonian cycle on $G[P] = G(P, \{e \in E \mid e \subseteq P\})$. Here, $G[P]$ is the sub-graph induced by $P$ consisting of all nodes in $P$ and all edges between them. Following [66], we represent a candidate solution as $S = (P, \pi)$. Let $p_{\pi_i}$ be the chosen node for cluster $V_{\pi_i}$, $1 \leq i \leq m$. Then the cost of a solution $S = (P, \pi)$ is given by

$$c(S) = c(p_{\pi_m}, p_{\pi_1}) + \sum_{i=1}^{m-1} c(p_{\pi_i}, p_{\pi_{i+1}}).$$

Different heuristic approaches for the GTSP have been presented in recent years [44, 52, 66, 98]. Moreover, two hierarchical approaches for solving this problem, the *Cluster-Based approach* and the *Node-Based approach*, are presented in [66]. Both approaches construct an overall solution based on an upper and lower level.

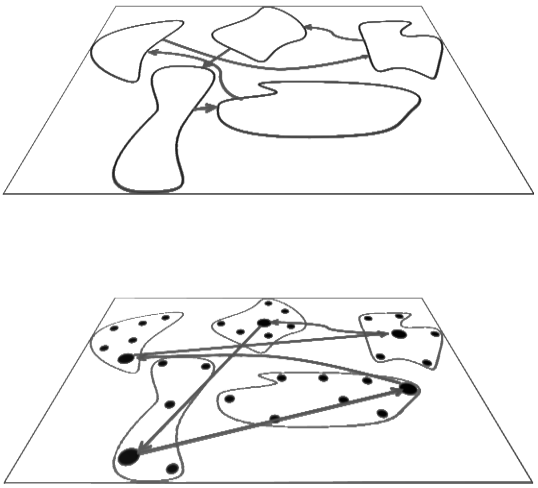

FIGURE 3.1: Cluster-Based approach for GTSP

The Cluster-Based approach, illustrated in Figure 3.1, uses a permutation on the clusters in the upper layer and finds the best node selection based on that permutation, on the lower layer. In other words, a tour on the clusters is suggested on the upper layer, and based on that tour, the lower layer selects one node for each cluster with the aim of minimizing the total cost. The permutation and the node selection form a solution for the whole problem. This solution is changed in an iterative fashion, starting in the the upper layer, i.e. the upper layer suggest another tour and the lower layer finds a (possibly) optimal node selection with respect to that tour. This new solution (new permutation together with the new node selection) is then compared to other solution/solutions that are found for the problem.

In contrast to the Cluster-Based approach, the Node-Based approach, illustrated in Figure 3.2, selects a spanning node for each cluster and then works on finding the best permutation of the chosen nodes. This means that the node selection is done in the upper layer in this approach, while the permutation is decided in the lower layer with respect to this node selection.

Hu and Raidl [66] have studied local search neighbourhood structures for the subproblems of GTSP, and have suggested that a variable neighbourhood search performs better that each of the local search approaches, as it performs a search on both neighbourhood structures. The two local search algorithms that they have studied, in addition to a variable neighbourhood search algorithm, are presented and investigated theoretically in Chapter 5.

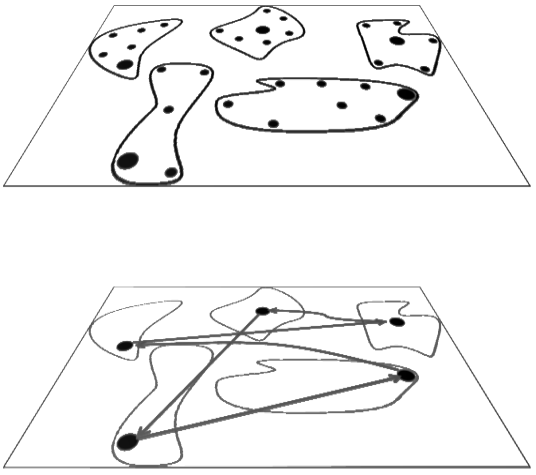

FIGURE 3.2: Node-Based approach for GTSP

## 3.5 Conclusion

In this chapter, we introduced the combinatorial optimization problems and presented the formal definition of two of the most well-known problems of this kind, namely the travelling salesman problem (TSP) and the vertex cover problem (VCP). The travelling salesman problem is one of the most studied combinatorial optimization problems for which a various number of methods have been suggested to solving the problem. In this chapter we only mentioned a few important methods that have achieved great success in dealing with TSP.

The vertex cover problem has also been studied well in the literature. We presented well-known approximation algorithms for the classical and weighted VCP. In order to explain the approximation algorithm for the weighted VCP, we needed the concept of duality. Therefore, we presented a brief introduction on LP problems and their dual form in Section 3.3.2.

Then we turned our attention to the concept of hierarchical optimization for combinatorial optimization problems, and defined the generalised travelling salesman problem (GTSP) as an example of problems for which hierarchical approaches can used.

# Chapter 4

# Methods of Algorithm Analysis in Bio-Inspired Computing

## 4.1 Introduction

The area of runtime analysis for bio-inspired computing techniques such as evolutionary algorithms and ant colony optimization started in 1992, with the first runtime analysis of an EA, given by Muhlenbein [86] . Since mid-1990s this area has provided many rigorous new insights into the working behaviour of bio-inspired computing methods for solving combinatorial optimization problems [6, 69, 91, 109, 110]. The computational complexity analysis of these algorithms studies the runtime behaviour with respect to the input size and plays a major role in their theoretical understanding. In this section we introduce some of the techniques that are of great importance in the field of runtime analysis for combinatorial optimization problems. Some of the strong techniques that can be applied to evolutionary algorithms, have been already used in the field of randomised algorithms [85]. Examples of these techniques are large deviation inequalities such as Markov's inequality and Chernoff bounds, and also the random walk problem and its variants. These techniques, in addition to the new techniques for analysing evolutionary algorithms such as fitness based partitions and drift analysis, are presented in this section.

## 4.2 Deviation Bounds

Large deviation inequalities are being used in analysing the behaviour of bio-inspired computing methods as well as randomized search algorithms. These inequalities bound

the probability that a random variable deviates from its expected value. Two of the mostly used techniques for finding these bounds, namely Markov's inequality and Chernoff bounds, are presented in this section.

### 4.2.1 Markov's Inequality

Markov's inequality is a simple and widely used deviation inequality which can be applied to any non-negative random variable $X$ with expected value of $E(X)$. As presented in Proposition A.4 of [91], this inequality says:

Let $X$ be a random variable which can take non-negative values. Then for all $k \in \mathbb{R}^+$,

$$\text{Prob}\left(X \geq k \cdot E(X)\right) \leq \frac{1}{k}.$$

Here we present an example of how this inequality can be used. Let $X$ be the time that is required by a randomised algorithm to perform a job. For example, let the algorithm select one item among $n^2$ items uniformly at random at each step, where $n$ denotes the input size. Also let $X$ be the number of steps that is required by the algorithm to select one specific item among those items. We here want to use Markov's inequality to show that the probability of $X > n^3$, is bounded by $e^{-\Omega(n)}$, i. e. with high probability, the item is selected by the algorithm in at most $n^3$ steps.

Since the selection is performed uniformly at random, the expected value of $X$ is $E(X) = n^2$. Considering $2n^2$ steps, from Markov's inequality we get:

$$\text{Prob}\left(X \geq 2 \cdot n^2\right) \leq \frac{1}{2}$$

Now consider $\frac{n}{2}$ phases of $2n^2$ steps, making a total of $n^3$ steps. The probability that the item is not selected in none of these phases is at most $(\frac{1}{2})^{n/2} = e^{-\Omega(n)}$.

### 4.2.2 Chernoff Bounds

Chernoff bounds gives exponentially decreasing bounds on the probability that the sum of some independent random variables deviate from their expected value. Comparing to Markov's inequality, it is a sharper bound but it requires the random variables to be independent. The following is the formulation of Chernoff bounds, presented in Proposition A.5 in [91].

Let $X_1, X_2, \cdots, X_n$ be independent Poisson trials such that $\text{Prob}(X_i = 1) = p_i$ for $1 \leq i \leq n$, where $0 < p_i < 1$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = E(X) = \sum_{i=1}^{n} p_i$. Then the

following inequalities hold.

$$\text{Prob}(X \geq (1+\delta)\mu) \leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu} \qquad \delta > 0$$

$$\text{Prob}(X \geq (1+\delta)\mu) \leq e^{-\mu\delta^2/3} \qquad 0 < \delta \leq 1$$

$$\text{Prob}(X \leq (1-\delta)\mu) \leq e^{-\mu\delta^2/2} \qquad 0 < \delta \leq 1$$

Since the variables $X_i$ have to be independent Poisson trials, Chernoff bounds are usually used for initial solutions that are formed of a number of random independent selections, and find the probability that these solutions have a specific feature. For example, consider an initial solution that is a bit-string of size $n$, and each bit is independently set to 0 or 1 uniformly at random. Let $X_i$ denote the value of bit $i$ and $X = \sum_{i=1}^{n} X_i$ be the number of ones in the bit-sting. Observe that $E(X) = \sum_{i=1}^{n} 1/2 = n/2$. Using Chernoff bounds with parameter $\delta = 1/2$, we can find the probability that the initial solution has at least $\frac{3n}{4}$ 1-bits as:

$$\text{Prob}\left(X \geq (1+1/2)\frac{n}{2}\right) \leq e^{-(\frac{n}{2})(\frac{1}{2})^2/3} = e^{-n/24}$$

Therefore, we can claim that with high probability, the number of 1-bits in the initial solution is less than $3n/4$.

## 4.3 Fitness Based Partitions

For this simple method, we assume that the considered algorithm is a stochastic search algorithm working with one solution that produces one offspring in each iteration. RLS and (1+1) EA that we have described in Sections 2.2.1 and 2.3.1 are examples of these kind of algorithms. Let $S$ be the search space and $f : S \to \mathbb{R}$ be the fitness function that should be maximised. Also, let $S$ be partitioned into disjoint sets $A_1, \cdots, A_p$ such that for all solutions $x \in A_i$ and $y \in A_j$ where $i < j$, it holds that $f(x) < f(y)$. Moreover, let $A_p$ contain only optimal search points. For a search point $x \in A_i$, the probability that in the next step a solution $y \in A_{i+1} \cup \cdots \cup A_p$ is produced, is denoted by $p(x)$. Also, the smallest probability of producing a solution with a higher partition number is denoted by $p_i = \min_{x \in A_i} p(x)$. With the assumptions described above, it is proved in Lemma 4.1 of [91] (presented here as Lemma 4.1), that the expected optimization time is upper bounded by $\sum_{i=1}^{p}(1/p_i)$. The idea of the proof is that from a solution in $A_i$, the expected time of moving to a partition $A_j$ where $j > i$, is at most $1/p_i$, as $p_i$ is the smallest probability of such a move. Since there are a total of $p$ partitions, $\sum_{i=1}^{p}(1/p_i)$ gives an upper bound on the expected time of reaching $A_p$.

**Lemma 4.1** (Lemma 4.1 of [91]). *The expected optimization time of a stochastic search algorithm that works at each time step with a population of size 1 and produces at each time step a new solution from the current solution is upper bounded by* $\sum_{i=1}^{p}(1/p_i)$.

To use this method for finding a good upper bound, a partitioning of the search space needs to be suggested such that for the current search point, there is a good probability of leaving the current partition and producing a search point in a better one. Moreover, there should not be too many partitions. To illustrate how to use this method, we here present a simple example. Consider a (1+1) EA that tries to find a matching on a given graph $G = (N, E)$, starting with a solution that is chosen uniformly at random. Let a solution $x$ be represented by $(x_1, x_2, \ldots, x_m)$, a bit-string of size $m = |E|$, where $x_i = 1$ (or $x_i = 0$) indicates that the edge $e_i$ is selected (or not selected) to be in the matching set. Also, let the fitness function be

$$f(x) = m - \left|\left\{e_i \mid x_i = 1 \ \wedge \ (\exists j \mid x_j = 1 \ \wedge \ e_i \cap e_j \neq \emptyset)\right\}\right|$$

i. e. $m$ minus the number of selected edges that share a node with another selected edge. Note that the fitness function should be maximized and the maximum value it can take is $m$ which denotes a matching. Also note that the maximum number of selected edges that share a node is upper bounded by $m$; therefore, the fitness function can only take discrete values between 0 and $m$. With partitioning the search space into $p = m + 1$ partitions $A_0, \cdots, A_p$ where $A_i$ consists of all solutions $x$ with $f(x) = i$, we here use the method of fitness based partitions to show that the expected optimization time of (1+1) EA for this problem is upper bounded by $O(n \cdot m)$.

A solution in partition $A_i, i < m$ has at least one edge that shares nodes with other edges, and can move to a partition $A_j, j > i$ if that the edge is deselected from the matching set. The probability of such an improvement is at least $1/(en)$ at each step. Therefore, the expected time until an improvement happens is upper bounded by $en$. Summing up the times for different partitions until $A_m$, the expected optimization time is $O(n \cdot m)$.

For analysing algorithms with larger population, an individual with the highest partition number in the population can be considered and the time until this individual reaches the optimal partition can be analysed.

**Algorithm 6:** Random Walk [91]

```
Start at a vertex v ∈ V;
repeat
    Choose a neighbor w of v in G uniformly at random;
    Set v := w
until stop;
```

## 4.4 Random Walk and the Gambler's Ruin Theorem

In the following, the random walk problem on a given graph is described and one of its important theorems is presented. The classical results on the random walk problem can be used for analysing stochastic search algorithms, particularly when dealing with plateau functions [91]. Moreover, in this section, a closely related theorem, known as the gambler's ruin theorem is explained which is used in proving exponential lower bounds, when the problem deals with unfair random walks.

### 4.4.1 Fair Random Walk

Given a connected graph $G = (V, E)$, the random walk problem is to start at a vertex $v \in V$ and in each step move to a neighbour of the current vertex that is chosen uniformly at random among all neighbours. Algorithm 6 describes this procedure.

The *cover time* of a random walk on its graph is the number of steps until all vertices have been visited at least once. The following theorem, obtained by Aleliunas et al. [2], gives an upper bound for the cover time of a random walk on a graph.

**Theorem 4.2.** *Given an undirected connected graph $G = (V, E)$ with $n$ vertices and $m$ edges, the cover time is upper bounded by $2|E|(|V| - 1)$.*

### 4.4.2 The Gambler's Ruin Theorem

The gambler's ruin theorem, introduced by Feller [43], is closely related to the discussion of random walks, but this time the probability of selecting a neighbour is not necessarily *fair*. The game is formulated as a Markov process on a space with $z+1$ states $s_i, 0 \leq i \leq z$, where $s_0$ and $s_z$ are absorbing states. From any state $s_i, 1 \leq i \leq b-1$, it is only possible to move to states $s_{i-1}$ and $s_{i+1}$, and the probability of these moves are denoted by $p$ and $q = 1 - p$, respectively. These transition probabilities are the same for all non-absorbing states. Starting in state $s_x, 1 \leq x \leq z-1$, the goal is to find the

absorbing state that is eventually reached. Interpreting the whole process as a gambling game where $x$ is the capital of a gambler and $z$ is the capital of the bank, the goal is to find the probability that the gambler wins. Here the gambler either wins or loses one unit of money in each step with a certain probability until either the bank or the gambler is ruined. We present a variant of the gambler's ruin theorem (Theorem 4.3) presented in [56].

**Theorem 4.3** (Gambler's Ruin Theorem)**.** *[56]*
*Let $p$ be the probability of winning one dollar and $q = 1 - p$ be the probability of loosing one dollar in a single bet and let $\delta = q/p$. Starting with $x$ dollars, the probability of reaching $z > x$ dollars before attaining zero dollars is*

$$P_x = \frac{\delta^x - 1}{\delta^z - 1}$$

If the initial capital of the gambler is low compared to the capital of the bank and the probabilities of the game are in favour of the bank, then the probability of the gambler's ruin is high. This theorem can be used for analysing stochastic search algorithms that tend to move towards a local optimum or a state that is hard to leave. In these situations, the Gambler's Ruin Theorem, as can be observed from the above expression, helps in proving an exponential lower bound, with respect to the parameter $z$, on the optimization time.

## 4.5 Drift Analysis

One of the most important tools for proving upper and lower bounds on the expected optimization times of evolutionary algorithms is *drift analysis*, introduced by He and Yao [57]. This method uses an auxiliary potential function to track the behaviour of the algorithm. In general, to use this method, one has to find the appropriate auxiliary function which its value is expected to be improved by at least a constant amount at each step of the algorithm. This method, which is known as *the additive drift analysis*, is explained in Section 4.5.1. While this method is very powerful and has gained great success, a variation of it is recently introduced in [34], that is quite straight forward to use, when the auxiliary function is improved by a portion of its current value at each step. This version of drift analysis is called *maltiplicative drift analysis* and is described in Section 4.5.2. The other drift theorem that we describe in this chapter is *the simplified drift theorem* [93, 94] (or the negative drift analysis) which is presented in Section 4.5.3. Considering a random variable with a positive expected improvement in an interval,

this theorem shows that with high probability, the whole interval is not passed towards the lower limit of the interval in an exponential time.

### 4.5.1 Additive Drift Analysis

Here we present a definition of *drift* as well as the first formal drift theorem, introduced by He and Yao [57]. The definition of the additive drift theorem below, is the version presented in [80] is which the discrete search space and the Markov property of the original definition are removed for simplicity.

**Definition 4.4** (**Drift** [80])**.** Consider a non-negative random variable $X_t, t \geq 0$ and a natural filtration $F_t = (X_0, \cdots, X_t)$, i.e. the information available up to time $t$. The expected one-step change $\delta_t = E(X_t - X_{t+1}|F_t)$ for $t > 0$ is called *drift*.

**Definition 4.5** (**Additive Drift** [80], following He and Yao [57])**.** Let $(X_t)_{t\geq 0}$, be a stochastic process over some bounded state space $S \subseteq \mathbb{R}_0^+$. Assume that $E(T_0|X_0) < \infty$ where $T_a = \min\{t|X_t \leq a\}$ is the first hitting time for threshold $a \geq 0$. Then:

- if $E(X_t - X_{t+1}|F_t; X_t > 0) \geq \delta_u$ then $E(T_0|X_0) \leq \frac{X_0}{\delta_u}$
- if $E(X_t - X_{t+1}|F_t) \leq \delta_l$ then $E(T_0|X_0) \geq \frac{X_0}{\delta_l}$

By applying the law of total expectation, the first statement implies $E(T_0) \leq \frac{E(X_0)}{\delta_u}$ and analogously for the second statement.

### 4.5.2 Multiplicative Drift Analysis

In the following theorem, the value of the random variable that is studied over time, $X^{(t)}$, is improved by a portion of its current value at each step. Using this theorem, runtime bounds have been found for evolutionary algorithms on several combinatorial optimization problems such as minimum spanning tree and shortest path problem [34].

**Theorem 4.6** (Multiplicative Drift [34])**.** *Let $S \subseteq \mathbb{R}$ be a finite set of positive numbers with minimum $s_{min}$. Let $\{X^{(t)}\}_{t\in\mathbb{N}}$ be a sequence of random variables over $S \cup \{0\}$. Let $T$ be the random variable that denotes the first point in time $t \in \mathbb{N}$ for which $X^{(t)} = 0$. Suppose that there exists a real number $\delta > 0$ that*

$$E\left[X^{(t)} - X^{(t+1)} \mid X^{(t)} = s\right] \geq \delta s$$

*holds for all $s \in S$ with $\text{Prob}[X^{(t)} = s] > 0$.*

*Then for all* $s_0 \in S$ *with* $\text{Prob}[X^{(0)} = s_0] > 0$, *we have*

$$E[T|X^{(0)} = s_0] \leq \frac{1 + \ln(s_0/s_{min})}{\delta}$$

### 4.5.3 Negative Drift Analysis

Consider a random variable $X_t, t \geq 0$ with positive values that is changed in a stochastic process. Also consider an interval of $[a, b], a \geq 0$. The following theorem shows that the lower limit of the interval is not reached by $X_t$ with high probability, if the starting point is above $b$, the average drift of the value of the random variable is positive, and the probability of having big changes on it is small.

The simplified drift theorem (Theorem 4.7) is presented in [93, 94], but the idea behind it goes back to [53]. In this theorem, $F_t$ denotes a filtration on states.

**Theorem 4.7** (Simplified Drift Theorem [94]). *Let* $X_t$ , $t \geq 0$, *be real-valued random variables describing a stochastic process over some state space. Suppose there exist an interval* $[a, b] \subseteq \mathbb{R}$, *two constants* $\delta, \varepsilon > 0$ *and, possibly depending on* $l := b - a$ , *a function* $r(l)$ *satisfying* $1 \leq r(l) = o(l/\log(l))$ *such that for all* $t \geq 0$ *the following two conditions hold:*

1. $E[X_{t+1} - X_t \mid F_t \wedge a < X_t < b] \geq \varepsilon$,

2. $\text{Prob}(|X_{t+1} - X_t| \geq j \mid F_t \wedge a < X_t) \leq \frac{r(l)}{(1+\delta)^j}$ *for* $j \in \mathbb{N}$.

*Then there is a constant* $c^* > 0$ *such that for* $T^* := min\{t \geq 0 : X_t \leq a | F_t \wedge X_0 \geq b\}$ *it holds* $\text{Prob}(T^* \leq 2^{c^* l/r(l)}) = 2^{-\Omega(l/r(l))}$.

## 4.6 Parameterized Complexity Analysis

In recent years, the parameterized analysis of bio-inspired computing has gained additional interest [37, 76, 77, 88, 111–113]. Here the runtime of bio-inspired computing is studied in dependence of the input size and some additional parameters which measure the hardness of a given instance, such as the solution size and structural parameters of the given input. This helps us gain an understanding on the problem hardness and find the parameters of a given NP-hard optimisation problem that make it hard or easy to solve by randomised search methods. Parameterized complexity analysis has been carried out for some NP-hard problems such as the computation of maximum leaf spanning trees [76], vertex cover problem [77], makespan scheduling [112] and the Euclidean travelling salesman problem [88, 111].

There are general algorithmic techniques for solving parameterized problems such as kernelization, integer linear programming (ILP), the method of bounded search trees, color coding, and dynamic programming on tree decompositions. Here we only give an introduction to kernelization and integer linear programming techniques and provide examples of how they can be used in parameterized analysis (Sections 4.6.2 and **??** respectively). We start with definition of fixed-parameter tractability (FPT) which is an important concept in the area of parameterized complexity analysis.

For a deeper review on parameterized complexity, refer to the textbook of Downey and Fellows [37] and Cygan et al. [20], both of which provide a comprehensive overview of important state of the art techniques in this field. Also, the book of Flum and Grohe [45] is suggested for an overview of the area, with a focus in complexity and hierarchies of intractable parameterized complexity classes.

### 4.6.1 Fixed-Parameter Tractable Algorithms

Assuming that $k$ is a relevant secondary measurement that encapsulates some aspect of the input instance, i. e. a structural parameter of a combinatorial problem, an algorithm is a fixed-parameter algorithm or a fixed-parameter tractable (FPT) algorithm if the expected runtime of the algorithm is bounded from above by $f(k).n^c$, where $n$ is the input size, $c$ is a constant independent of $n$ and $k$, and $f(k)$ is a function depending on $k$ only. An evolutionary algorithm that is an FPT is called a fixed-parameter evolutionary algorithm.

For giving a more formal definition of FPT problems and algorithms, we denote a parameterized problem by a language $L \subseteq \Sigma^* \times \Sigma^*$, where $\Sigma$ is a fixed, finite alphabet. For most parameterized problems, the parameter is a positive integer and the language can be represented by $L \subseteq \Sigma^* \times \mathbb{N}$. Given that in an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, $k$ denotes the parameter, the formal definition of an FPT problem is as the following.

**Definition 4.8** (Fixed-Parameter Tractable (FPT) Problem [20])**.** A parameterized problem $L = \Sigma^* \times \mathbb{N}$ is called fixed- parameter tractable (FPT) if there exists an algorithm $A$ (called a fixed-parameter algorithm), a computable function $f : \mathbb{N} \to \mathbb{N}$, and a constant $c$ such that, given $(x, k) \in \Sigma^* \times \mathbb{N}$, the algorithm $A$ correctly decides whether $(x, k) \in L$ in time bounded by $f(k) \cdot |(x, k)|^c$.

### 4.6.2 Kernelization

Kernelization is the process of reducing the size of an NP-hard problem instance. This process aims at solving the easy parts of the problem efficiently and find an equivalent

smaller instance which may still need exponential time to be solved. The output of a kernelization algorithm is called the reduced equivalent instance or a kernel. The kernelization process needs to be done in polynomial time with respect to the size of the original problem, and a solution for this problem needs to be derived, from a solution that is later found for the kernel.

Denoting a parameterized problem by a language $L \subseteq \Sigma^* \times \Sigma^*$, the formal definition of kernelization is as the following.

**Definition 4.9** (Kernelization [37])**.** Let $L \subseteq \Sigma^* \times \Sigma^*$ be a parameterized language. A reduction to a problem kernel, or kernelization, replaces an instance $(x, k)$ by a reduced instance $(x', k')$, called a problem kernel, such that

- $k' \leq k$,
- $|x'| \leq g(k)$, for some function $g$ depending only on $k$, and
- $(x, k) \in L$ if and only if $(x', k') \in L$.

The reduction from $(x, k)$ to $(x', k')$ must be computable in time polynomial in $|x| + |k|$.

If there exists a kernelization algorithm for a problem, then an instance $(x, k)$ can be reduced to an instance $(x', k')$ with a size of at most $g(k) + k$. Therefore, if there is an algorithm to solve the instance $(x', k')$, even of it requires exponential time with respect to the instance size, the run time of solving the instance $(x', k')$ is a function of $k$ only. Recalling Definition 4.8, it is clear that the problem is FPT with parameter $k$.

On the other hand, it also holds that if a parameterized problem is FPT, then it admits a kernelization algorithm (Lemma 2.2 in [20]). This implies that a decidable problem admits a kernel if and only if it is an FPT problem. The idea behind this lemma is that since the problem is FPT, there exists an algorithm $A$ that solves an instance $(x, k)$ in time $f(k).|x|^c$. If the the actual run time is less than or equal to $|x|^{c+1}$, then $A$ can be assumed as a kernelization algorithm that returns the result in polynomial time. Otherwise, $f(k).|x|^c \geq |x|^{c+1}$, which implies $f(k) \geq |I|$. Therefore, the size of the instance $(x, k)$ is bounded by $f(k) + k$, and the the whole instance can be returned as the kernel. For the detailed proof of this lemma refer to [20].

As a simple example for kernelization, consider the decidable version of the classical vertex cover problem, in which a parameter $k$ is given in addition to the input graph, and the goal is to decide whether a vertex cover of size at most $k$ can be found for the given instance. We here show that Algorithm 7 is a kernelization algorithm for this problem.

**Algorithm 7:** Kernelization for the vertex cover problem

```
Given is the parameter k and the graph G = (V, E);
Initialize k' ← k and G' = (V', E') ← G;
Initialize C ← ∅;
repeat
    Rule 1: delete isolated vertices of G';
    Rule 2: if there exists a vertex v with degree greater than k' then
        C ← C ∪ {v};
        k' ← k' − 1;
        V' ← V' \ v;
        for ∀e ∈ E' where v ∈ e do
            E' ← E' \ e;
until neither Rule 1 nor Rule 2 can be applied;
if |V'| > k'^2 or |E'| > k'^2 then
    There is no vertex cover of size at most k for G;
else
    Return k' and G' as the kernel to be investigated;
```

Observe that Algorithm 7 goes through its repeat loop at most $|V|$ times and the time that each iteration requires is bounded by $O(|V| \cdot |E|)$. Therefore, the algorithm is performed in polynomial time with respect to the input size. Since $k'$ is initialized to $k$ and can only be decreased in the process of the algorithm, we have $k' \leq k$.

Rule 1 removes the isolated vertices since they do not play a role in making a cover. Moreover, Rule 2 is applied because a vertex of degree greater than $k'$ has to be in the cover set; otherwise, the cover set would need to include all adjacent vertices of that vertex which exceed its maximum allowable size. After each iteration of the repeat loop, a vertex cover of size $k'$ for $G'$, in addition to the set of vertices $C$ found so far by the algorithm, forms a vertex cover of size $k$ for the original graph $G$. In other words, there is a vertex cover of size at most $k'$ for $G'$ if and only if there is a vertex cover of size at most $k$ for $G$.

The main repeat loop terminates when there are no isolated vertices and no vertices have a degree greater than $k'$. At this point, if there exists a vertex cover of size at most $k'$ for $G'$, then the number of edges in $G'$ is upper bounded by $k'^2$, because this is the maximum number of edges that can be covered by a set of $k'$ vertices of degree at most $k'$. Also, since there are no isolated vertices in $G'$, the number of vertices is upper bounded by $k'^2 + 1$. This implies that if $|V'| > k'^2 + 1$ or $|E'| > k'^2$, then there is no vertex cover of size at most $k'$ for $G'$. Otherwise, the size of the graph $G'$ is bounded by a function of $k'$ (and also $k$; since we have $k' < k$), and $(G', k')$ can be considered as the kernel for instance $(G, k)$.

### 4.6.3 Integer Linear Programming Technique

Using Integer Linear Programming (ILP) is another technique for solving parameterized problems. Many combinatorial problems can be presented in the form of an ILP, where a set of integer-valued variables is given together with a set of linear inequalities (constraints) and a linear objective function. As we explained in Section 3.3.2, the goal of an ILP is to find integer values of the variables that satisfies all constraints, and minimizes or maximizes the value of the objective function.

While an ILP problem can be NP-hard, the relaxed version of that problem is a linear programming problem which can be solved in polynomial time. The information obtained by solving a relaxed version of some of the NP-hard combinatorial problems can be used is finding solutions for the ILP version of the problem. An example of this, is the use of LP to find a kernel for the vertex cover problem on a given graph $G = (V, E)$. The ILP formulation of the vertex cover problem and the relaxation on that are explained in Section 3.3.2. Recall that a solution of the ILP problem is represented by a bitstring $x = (x_1, \cdots, x_n)$, and the constraint $x_i \in \{0, 1\}$ is relaxed to $x_i \in [0, 1]$ in the relaxed version of the problem. Using an optimal solution of the LP problem $x$, consider the following partitioning on $V$.

- $V_0 = \{v_i \in V \mid x_i < 1/2, i \in \{1, \cdots n\}\}$
- $V_{\frac{1}{2}} = \{v_i \in V \mid x_i = 1/2, i \in \{1, \cdots n\}\}$
- $V_1 = \{v_i \in V \mid x_i > 1/2, i \in \{1, \cdots n\}\}$

With this partitioning, the following theorem is proved [20].

**Theorem 4.10** (Nemhauser-Trotter theorem as presented in [20])**.** *There is a minimum vertex cover $S$ of $G$ such that*

$$V_1 \subseteq S \subseteq V_1 \cup V_{\frac{1}{2}}$$

The idea of the proof is to consider a minimum vertex cover $S^*$ of $G$, define $S = (S^* \setminus V_0) \cup V_1$, and prove that $S$ is also a minimum vertex cover of $G$. It is easy to show that $S$ is a vertex cover, because every vertex in $V_0$ can only have a neighbour in $V_1$. What remains is to prove that $S$ is a *minimum* vertex cover. In order to prove this, one can assume the contrary, i.e. $|S| > |S^*|$ and use the values of $x_i$ to obtain another feasible solution for the relaxed version of the problem, with a lower cost than $x$. This contradicts with $x$ being an optimal solution of the LP problem. For the details of the proof, refer to Theorem 2.19 of [20].

Theorem 4.10 implies that if we include all vertices of $V_1$ in the cover set, and define $k'$ and $G' = (V', E')$ as $k' = k - |V_1|$ and $V' = V_{\frac{1}{2}},\ E' = \{e \in E \mid e \subseteq V'\}$, then we obtain a (possibly) smaller equivalent instance of the problem $(G', k')$. Note that if $\sum_{i=1}^{n} x_i > k$, then the answer to the original problem $(G, k)$ is no, i. e. there is no vertex cover of size at most $k$ for $G$. Therefore, we only need to consider the case where $\sum_{i=1}^{n} x_i \leq k$. This implies that $|V_{\frac{1}{2}}| \leq 2k$. In other words, the size of the graph $G'$ is a function of the parameter $k$; therefore, the vertex cover problem $(G, k)$ is an FPT problem with respect to parameter $k$.

## 4.7 Conclusion

In this chapter we presented some of the strong techniques that are widely used in the field of runtime analysis for combinatorial optimization problems. We started by deviation bounds such as Markov's inequality and the Chernoff bounds which help us in finding the maximum probability that a random variable deviates from its expected value. Then we presented the fitness-based partition technique where the proper partitioning of the solution space plays an important role in the final results that is obtained by this technique. We also presented the fair random walk problem and the Gambler's Ruin Theorem which can be used in proving upper bounds on the expected time and the probability of reaching a state, respectively. Moreover, we included important theorems of the field of drift analysis, which provide strong tool in finding upper and lower bounds on the expected optimization times of stochastic search algorithms. At the end of this chapter, we presented a brief description on parameterized complexity analysis and FPT algorithms.

# Chapter 5

# Local Search and the Generalized Travelling Salesman Problem

## 5.1 Introduction

In this chapter we present the theoretical understanding of local search methods for the generalized travelling salesman problem (GTSP). The problem is given by a set of cities with distances between them. Furthermore, the cities are divided into clusters and the goal is to find a tour of minimal cost that visits one city from each cluster exactly once. The formal problem definition can be found in Section 3.4.1. Here we investigate the two hierarchical approaches for solving the GTSP presented in [66] (Section 3.4.1) from a theoretical perspective. Our aim is to show situations where one of the approaches gets stuck in a local optimum and the other approach is able to perform well and achieve an optimal solution. This gives a deeper insight into the working principles of these two common approaches and highlights their complementary abilities. Furthermore, we present an instance where both local search approaches are not able to achieve an optimal solution, but a combination of them into a variable-neighbourhood search reaches an optimal solution. To gain these structural insights, our instances should be simple enough for theoretical treatment. As we are considering hierarchical approaches working with two solution layers, it is very difficult to argue in general about the run of metaheuristics on these problems. The only runtime analysis in the context of parameterized complexity for a hierarchical optimization problem that we are aware of, is the analysis of simple evolutionary algorithms for the generalized minimum spanning tree problem [19], which shows that evolutionary algorithms using a cluster-based approach perform well for this problem if the number of clusters is small.

The contents of this chapter are based on a GECCO conference paper [101] and a paper submitted to a journal [102]. Section 5.2 of this chapter introduces the algorithms that are subject to our investigations. In Section 5.3, we introduce a hard instance for the Cluster-Based approach which is easy to solve for Node-Based approach. Section 5.4 includes an instance easy for Cluster-Based approach and difficult for Node-Based approach and Section 5.5 introduces the third instance which is difficult for both of them but an algorithm that combines the two approaches can solve it easily. Existence of such an instance strengthens the idea that combining the two approaches is beneficial due to searching different neighbourhoods of the problem. Finally, we finish with some concluding remarks in Section 9.6.

## 5.2 Local Search Algorithms for the Generalised Travelling Salesman Problem

In this section we describe the local search algorithms based on the two hierarchical approaches of solving GTSP, the Cluster-Based approach and the Node-Based approach (Section 3.4.1), and also the variable neighbourhood search of [66].

### 5.2.1 Cluster-Based Local Search

In the Cluster-Based approach, constructing the permutation of clusters constitutes the upper layer and the node selection is done in the lower layer [66]. Let $\pi = (\pi_1, \cdots, \pi_m)$ be a permutation of the $m$ clusters and $P = \{p_1, p_2, \cdots, p_m\}$ be the set of selected nodes. Also, let $p_{\pi_i}$ be the chosen node for cluster $V_{\pi_i}$, $1 \leq i \leq m$. Then, as mentioned in Section 3.4.1, the cost of a solution $S = (P, \pi)$ is

$$c(S) = c(p_{\pi_m}, p_{\pi_1}) + \sum_{i=1}^{m-1} c(p_{\pi_i}, p_{\pi_{i+1}}),$$

Furthermore, the 2-opt neighbourhood of $\pi$ is given by

$$N(\pi) \quad = \quad \{\pi' \mid \pi' = (\pi_1, \cdots, \pi_{i-1}, \pi_j, \pi_{j-1}, \cdots, \pi_i, \pi_{j+1}, \cdots, \pi_m), \quad 1 \leq i < j \leq m\}$$

The Cluster-Based local search (CBLS) working with this neighbourhood structure, given in Algorithm 8, starts with an initial permutation of clusters. At each step, a new permutation $\pi'$ is selected from the 2-opt neighbourhood of $\pi$, the current permutation of clusters. Then the lower layer finds the best spanning node set for that permutation,

**Algorithm 8:** Cluster-Based local search (CBLS)

```
1 Choose a permutation π = (π_1, ..., π_m);
2 Find the optimal set of spanning nodes P with respect to π to obtain the solution
  S = (P, π);
3 for π' ∈ N(π) do
4     Find an optimal set of nodes P' = {p'_1, ..., p'_m} with respect to π' to obtain the
      solution S' = (P', π');
5     if c(S') < c(S) then
6         S = S';
7         GO TO 3
```

and the new solution $S' = (P', \pi')$ replaces the old one if it is less costly. The algorithm terminates if no better solution can be found in the 2-opt neighbourhood of $\pi$.

The lower layer uses a shortest path algorithm to find the best spanning node set. Hu and Raidl [66] have applied an incremental bidirectional shortest path calculation for this purpose. The shortest path algorithm of [71] is another option, which is an improved version of dynamic programming algorithm of [44] for finding an optimal set of spanning nodes for a given permutation in time $O(n^3)$. Section 5.2.1.1 describes this algorithm which is presented in Algorithm 9.

#### 5.2.1.1 Cluster Optimisation Algorithm

This section describes the polynomial algorithm *Cluster Optimisation* (Algorithm 9) proposed initially by Fischetti et al [44] and improved by Karapetyan and Gutin [71]. Considering a given order of clusters, the algorithm finds the optimal set of nodes to visit for GTSP. The main idea of this algorithm is to represent the problem as a layered network in which each layer is a cluster of nodes and the last layer is a copy of the first cluster (Figure 5.1). This representation is possible since the cluster permutation is imposed by some other component.

At the beginning we assume that the first cluster has a fixed node to start the tour and from that node we move through the network layer by layer. In each layer we find the


FIGURE 5.1: Layered network of nodes

**Algorithm 9:** Cluster Optimization [71]

Get tour $T = (T_1, T_2, \cdots, T_m)$ from input;
Let $\tau_i = Cluster(T_i)$, $1 \leq i \leq m$;
**for** $v \in \tau_1$ *and* $r \in \tau_2$ **do**
    Set the shortest path from $v$ to $r$: $p_{v,r} \leftarrow (v, r)$;
**for** $i \in \{3, \cdot, m\}$ **do**
    **for** $v \in \tau_1$ *and* $r \in \tau_i$ **do**
        $p_{v,r} \leftarrow p_{v,u} \cup (u, r)$ where $u \in \tau_{i-1}$ is selected to minimize $w(p_{v,u} \cup (u, r))$
return $p_{v,r} \cup (r, v)$ where $v \in \tau_1$ and $r \in \tau_m$ are selected to minimize $w(p_{v,r} \cup (r, v))$

minimum cost to reach each of the nodes. When we are done with the last layer, the shortest path from the initial node to the corresponding node in the last layer is found. In order to make sure that no path is missing, we need to do the whole procedure for each of the nodes in the first cluster. As a result, the cardinality of the first cluster plays an important role in the eventual complexity of the algorithm. Therefore, the first cluster is better to be the smallest cluster in terms of number of the nodes. If we represent the total number of nodes, the minimum cardinality of clusters and the maximum cardinality of them by $n$, $\gamma$ and $s$ respectively, then the time complexity of the algorithm is $\gamma ns = O(n^3)$. Here we explain how this time bound is obtained. The minimum cost should be found for all of the nodes in the network which is $n$. To find the minimum cost at each node, every node at the previous layer should be considered and the sum of their cost and the weight of the edge between them and the current node should be calculated. Therefore, the number of nodes in the previous layer (which is at most $s$) is multiplied by $n$. Finally, we see $\gamma$ in the formula because the whole described procedure is repeated for every nodes of the first layer.

### 5.2.2 Node-Based Local Search

In the Node-Based approach [66], selection of the spanning nodes is done in the upper layer and the lower level consists of finding a shortest tour on the spanning nodes. Given a spanning nodes set $P$, in the nodes based local search algorithm, the upper layer performs a local search based on the node exchange neighbourhood $N'(P)$ that is defined as

$$N'(P) \quad = \quad \{P' \mid P' = \{p_1, \cdots, p_{i-1}, p'_i, p_{i+1}, \ldots, p_m\}, \quad p'_i \in V_i \setminus \{p_i\}, \quad 1 \leq i \leq m\}$$

Note that the lower level involves solving the classical TSP; therefore, it poses in general an NP-hard problem on its own. For our theoretical investigations, we consider two algorithms NEN-LS (Node Exchange Neighbourhood local search) and NEN-LS*

**Algorithm 10:** Node Exchange Neighbourhood local search (NEN-LS)

1 Choose $P = \{p_1, p_2, \ldots, p_m\}, \quad p_i \in V_i$;
2 Let $\pi$ be the permutation of clusters obtained by performing a 2-opt local search on $G[P]$ and $S = (P, \pi)$ be the resulting solution;
3 **for** $P' \in N'(P)$ **do**
4 Let $\pi'$ be the permutation of clusters obtained from $\pi$ by performing a 2-opt local search on $G[P']$ and $S' = (P', \pi')$ be the resulting solution;
5 **if** $c(S') < c(S)$ **then**
6 $S = S'$;
7 GO TO 3

**Algorithm 11:** Node Exchange Neighbourhood local search* (NEN-LS*)

1 Choose $P = \{p_1, p_2, \cdots, p_m\}, \quad p_i \in V_i$;
2 Find a minimum-cost permutation $\pi$ for $G[P]$ and let $S = (P, \pi)$ be the resulting solution;
3 **for** $P' \in N'(P)$ **do**
4 Find a minimum-cost permutation $\pi'$ for $G[P']$ and let $S' = (P', \pi')$ be the resulting solution;
5 **if** $c(S') < c(S)$ **then**
6 $S = S'$;
7 GO TO 3

presented in Algorithm 10 and Algorithm 11 respectively. NEN-LS computes a permutation on the lower level using 2-opt local search and is therefore not guaranteed to reach an optimal permutation $\pi$ for a given spanning node set $P$. NEN-LS* uses an optimal solver to find an optimal permutation $\pi$ for a given spanning node set $P$. Such a permutation can be obtained in time $O(m^2 2^m)$ using dynamic programming [58] and is practical if the number of clusters is small. We use NEN-LS* and show where it gets stuck in local optima even if the travelling salesman problem on the lower level is solved to optimality.

NEN-LS and NEN-LS* start with a spanning node set $P$ and search for a good or optimal permutation with respect to $P$. Then each solution $P' \in N'(P)$ together with its permutation $\pi'$ is considered and $S' = (P', \pi')$ replaces the current solution $S = (P, \pi)$ if it is of smaller cost. The definition of cost of a solution $S$ is similar to that of Section 5.2.1. Both algorithms terminate if there is no improvement possible in the neighbourhood $N'(P)$ of the current solution $P$.

**Algorithm 12:** Variable Neighborhood Search (VNS)

```
1 Choose an initial solution S = (P, π);
2 l = 1;
3 while l ≤ 2 do
4     for S' ∈ N_l(S) do
5         if c(S') < c(S) then
6             S = S';
7             l = 1;
8             GO TO 3
9     l = l + 1
```

### 5.2.3 Variable Neighbourhood Local Search

Hu and Raidl [66] have also introduced the combination of the two approaches into a variable neighbourhood search algorithm and shown that this leads to a high performing algorithm for the GTSP. Two neighbourhood structures of CBLS and NEN-LS are used in their algorithm. For the combination of these two approaches, the Variable Neighbourhood Search (VNS) scheme is used with embedded Variable Neighbourhood Descent (VND) as proposed in [55]. The variable neighbourhood search of [66] uses the NEN-LS structure only when the algorithm is in a local optimum with respect to the CBLS structure. Motivated by their algorithm, we define another version of variable neighbourhood local search (Algorithm 12) which explores different neighbourhood structures until they stick to a local optimum.

Let $S = (P, \pi)$ be a solution to the GTSP. We define the two neighbourhoods $N_1$ and $N_2$ based on the 2-opt neighbourhood $N$ and the node exchange neighbourhood $N'$ as

- $N_1(S) = \{S' = (P', \pi') \mid \pi' \in N(\pi), P' = \text{optimal set of nodes with respect to } \pi'\}$
- $N_2(S) = \{S' = (P', \pi') \mid P' \in N'(P), \pi' = \text{order of clusters obtained by 2-opt from } \pi \text{ on } G[P']\}$

Combining the two local searches of Cluster-Based approach and Node-Based approach is done by alternating between $N_1$ and $N_2$. $N_1$ is the first neighbourhood to be searched. When a local optimum has been found with respect to $N_1$, then neighbourhood $N_2$ is being searched. The process continues by searching $N_1$ again, when a local optimum is found with respect to $N_2$.

## 5.3 Benefits of Cluster-Based Local Search

We now consider a situation where NEN-LS* finds it hard to obtain an optimal solution and CBLS with the same starting solution obtains an optimum in polynomial time. The instance $G_2 = (V, E)$ is illustrated in Figure 5.2. There are $m$ clusters where $m > 2$, and all the clusters contain only 2 nodes; one white and one black. We refer to the white and black nodes of cluster $i$, $1 \leq i \leq m$, by $v_{iW}$ and $v_{iB}$, respectively. We call cluster $V_1$ the costly cluster as edges connecting this cluster are more costly than edges connecting the other clusters. The edge set $E$ of this complete graph is partitioned into $4$ different types.

- Type $A$: Edges of this type have a weight of $1$. All connections between white nodes of different clusters except cluster $V_1$ are of this type.

$$A = \{\{v_{iW}, v_{jW}\} \mid 2 \leq i, j \leq m\}$$

- Type $B$: Edges of this type have a weight of 2. All connections between black nodes of different clusters are of this type.

$$B = \{\{v_{iB}, v_{jB}\} \mid 1 \leq i, j \leq m\}$$

- Type $C$: Edges of this type have a weight of $m$. All edges between white node of the costly cluster and white nodes of other clusters are of this type.

$$C = \{\{v_{1W}, v_{iW}\} \mid 2 \leq i \leq m\}$$

- Type $D$: Edges of this type have a weight of $m^2$. All other edges in this complete graph, which consist of all edges between a white and a black node, are of this type.

$$D = E \setminus \{A \cup B \cup C\} = \{\{v_{iW}, v_{jB}\} \mid 1 \leq i, j \leq m\}$$

We first claim that the optimal solution consists of only black nodes. Then we bring our main theorems on the runtime behaviour of solving this instance with the two mentioned approaches.

*Property* 5.1. For the graph $G_2$ any solution containing all black nodes is optimal.

*Proof.* A solution that contains only black nodes has $m$ edges of type $B$ and therefore total cost of $2m$.

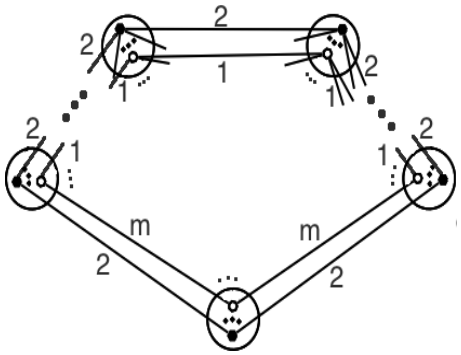


FIGURE 5.2: Graph $G_2$

Choosing a combination of black and white nodes implies a connection of type $D$ and therefore a solution of cost at least $m^2$. Choosing all white nodes implies 2 edges of cost $m$ connected to cluster $V_1$ and $m-2$ edges of cost 1. Hence, the total cost of such a solution is $2m + (m-2)$ which implies that a solution selecting all black nodes is optimal. □

We now show that CBLS always finds an optimal solution due to selecting an optimal spanning nodes in time $O(n^3)$.

**Theorem 5.2.** *Starting with an arbitrary permutation $\pi$, CBLS finds an optimal solution for $G_2$ by choosing the optimal spanning node set $P$ for $\pi$ in time $O(n^3)$.*

*Proof.* As mentioned in Property 5.1, visiting black nodes of the graph in any order is a globally optimal solution. For each permutation $\pi$ the optimal set of nodes is given by all black nodes and found when constructing the first spanning node set. Such a set $P$ is constructed in time $O(n^3)$ by the shortest path algorithm given in [71]. □

In contrast to the positive result for CBLS, NEN-LS* is extremely likely to get stuck in a local optimum if the initial spanning node set is chosen uniformly at random. Note, that NEN-LS* even uses an exact solver for the lower layer.

**Theorem 5.3.** *Starting with a spanning node set $P$ chosen uniformly at random, NEN-LS* gets stuck in a local optimum of $G_2$ with probability $1 - e^{-\Omega(n)}$.*

*Proof.* Selecting $P = \{p_1, \cdots, p_m\}$ uniformly at random, the expected number of white nodes is $\frac{n}{2}$. Using Chernoff bounds, the number of white nodes is at least $n/4$ with probability $1 - e^{-\Omega(n)}$. The same applies to the number of black nodes.

Since connecting white nodes to black nodes is costly, the lower layer selects a permutation which forms a chain of white nodes and a chain of black nodes connected to form a cycle by only two edges of type $D$.

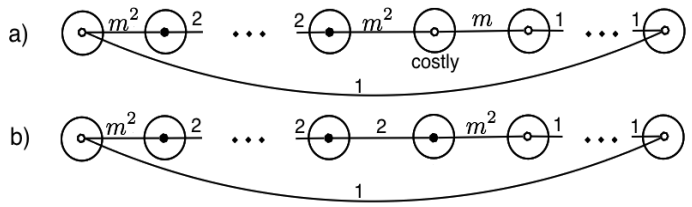


FIGURE 5.3: The initial solution for $G_2$ if a) A white node is selected for the costly cluster. b) A black node is selected for the costly cluster.

Let $p_1$ be the selected node of the costly cluster $V_1$. If $p_1$ is initially white, the lower layer places it at one border between the black chain and the white chain to avoid using one of the edges of type $C$. This situation is illustrated in Figure 5.3-a. If $p_1$ is initially black, then the initial solution would look like Figure 5.3-b, in which the costly cluster is placed somewhere in the black chain.

*Claim* 5.4. Starting with a random initial solution, with probability $1 - e^{-\Omega(n)}$ for all the clusters $V_i, 2 \leq i \leq m$; a change from black to white is improving while no change from white to black is improving.

*Proof.* As mentioned earlier, a random initial solution has both kinds of nodes with probability $1 - e^{-\Omega(n)}$; therefore, it contains a chain of black nodes and a chain of white nodes. Changing a black node $p_i, i \neq c$ to white results in shortening the chain of black nodes by removing an edge of type $B$ and cost 2, while the chain of white nodes gets longer by adding an edge of type $A$ and cost 1. The new solution is hence improved in terms of fitness and accepted by the algorithm. On the other hand, the opposite move increases the cost of the solution; therefore in a cluster $V_i, i \neq c$ a change from white to black cannot happen.

The number of selected white nodes for clusters $V_i, i \neq 1$ never decreases; therefore, at all time during the run of the algorithm we have both chains of black nodes and white nodes, until all the black nodes change to white. □

*Claim* 5.5. As long as there is at least one cluster $V_i$, $i \neq 1$ for which the black node is selected, a change from white to black is accepted for cluster $V_1$ and the opposite change is rejected.

*Proof.* Since there is at least one cluster $V_i, i \neq 1$, for which the black node is selected, we know that the current solution and the new solution both have a chain of black nodes and a chain of white nodes. If the white node of cluster $V_1$ is selected in the current solution, changing it to black shortens the chain of white nodes with removing the edge of type $C$ while increases the number of black nodes by adding an edge of type $B$. This move is accepted because the new solution is improved in terms of cost.

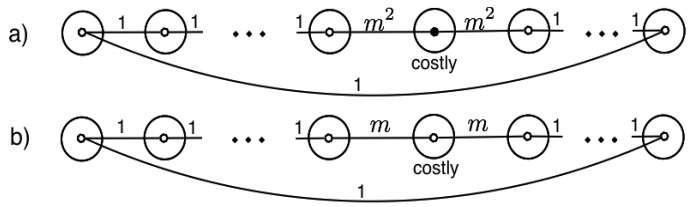


FIGURE 5.4: a) All other clusters change to white one by one. b)Local Optimum for $G_2$

The result is illustrated in Figure 5.3-b. Using similar arguments, if the black node of cluster $V_1$ is selected in the current solution, changing it to white is rejected because it increases the cost. □

Using Claim 5.4 we can conclude that all nodes $p_i, i \neq 1$ gradually are changed to white in NEN-LS. For $p_1$:

- If $p_1$ is initially black, it remains black until all other $p_i$s change to white. At this point $p_1$ is the only black node in the solution and is connected to two white nodes with edges of type $D$ and cost $m^2$ as illustrated in Figure 5.4-a. If it changes to white, these two edges are removed and two edges of type $C$ and cost $m$ are added to the solution (Figure 5.4-b). This change is accepted because two edges of cost $m$ are less costly than two edges of cost $m^2$.
- If $p_1$ is initially white,
  - If it happens to change to black, it remains black until all other $p_i$s change to white, at which point $p_1$ also changes to white.
  - If all other $p_i$s change to white before trying a black node for $p_1$, then it never changes to black.

This eventually results in a local optimum with all white nodes selected. The algorithm only needs to traverse the clusters on the upper layer only twice which gives $O(m)$ iterations on the upper layer for the algorithm to get stuck in a local optimum. In the first traversal, for all the clusters the white node will be selected except for the costly cluster $V_1$ for which the black node will be selected. In the second traversal, that only black node will also change to white.

This completes the proof of Theorem 5.3. □

## 5.4 Benefits of Node-Based Local Search

In this section, we present an instance of the problem that can not be solved by CBLS. In contrast to this, NEN-LS finds an optimal solution in polynomial time.

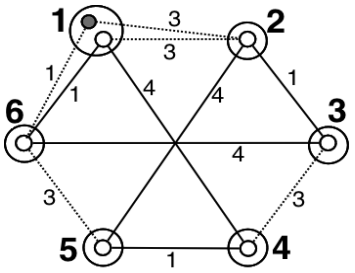


FIGURE 5.5: $G_1$, an easy instance for NEN-LS and a hard instance for CBLS

We consider the undirected complete graph, $G_1 = (V, E)$ which is illustrated in Figure 5.5. The graph has $n$ nodes and 6 clusters $V_i$, $1 \leq i \leq 6$. Cluster $V_1$ contains $n/12$ white and $n/12$ grey nodes. We denote by $V_{1W}$ the subset of white nodes and by $V_{1G}$ the subset of grey nodes of cluster $V_1$. Each other cluster $V_j$, $2 \leq j \leq 6$, consists of $n/6$ white nodes. The node set $V = \cup_{i=1}^{6} V_i$ of $G_1$ consists of nodes of all clusters. For simplicity, Figure 5.5 shows only one node for each group of similar nodes with similar edges in the picture. The edge set $E$ consists of 4 types of edges which we define in the following.

- Type $A$: Edges of this type have a cost of 1. All edges between clusters 2 and 3, and between clusters 4 and 5 and also between clusters 6 and 1, are of this type.

  $$A = \{\{v_i, v_j\} \mid (v_i \in V_{1W} \cup V_{1G} \wedge v_j \in V_6) \vee (v_i \in V_2 \wedge v_j \in V_3) \vee (v_i \in V_4 \wedge v_j \in V_5)\}$$

- Type $B$: Edges of this type have a cost of 3. All edges connecting the nodes of cluster 1 to cluster 2 are of this type. So are the edges that connect nodes of cluster 3 to 4 and cluster 5 to 6.

  $$B = \{\{v_i, v_j\} \mid (v_i \in V_{1W} \cup V_{1G} \wedge v_j \in V_2) \vee (v_i \in V_3 \wedge v_j \in V_4) \vee (v_i \in V_5 \wedge v_j \in V_6)\}$$

- Type $C$: Edges of this type have a cost of 4. All edges between nodes of cluster 2 and 5 and also between clusters 3 and 6 are of this type. All edges that connect white nodes of the first cluster to nodes of the forth cluster are also of this type.

  $$C = \{\{v_i, v_j\} \mid (v_i \in V_{1W} \wedge v_j \in V_4) \vee (v_i \in V_2 \wedge v_j \in V_5) \vee (v_i \in V_3 \wedge v_j \in V_6)\}$$

- Type $D$: Edges of this type have a large cost of $100$. All edges other than those of type $A$ or $B$ or $C$ in this complete graph, including the edges between grey nodes of the first cluster and the nodes of the forth cluster, are of Type $D$.

$$D = E \setminus \{A \cup B \cup C\}$$

We say that a permutation $\pi = (\pi(1), \ldots, \pi(n))$ visits the cities in consecutive order iff $\pi(i+1) = (\pi(i) \bmod n) + 1$, $1 \leq i \leq n$ and say that $\pi = (\pi(1), \ldots, \pi(n))$ visits the cities in reverse-consecutive order iff $\pi(i) = (\pi(i+1) \bmod n) + 1$, $1 \leq i \leq n$.

*Property* 5.6. For the instance $G_1$, each solution visiting the clusters in consecutive or reverse-consecutive order is optimal.

*Proof.* The graph consists of $6$ clusters which implies that $6$ edges are needed for a tour. The least costly edges are of type $A$, which are available only between 3 pairs of clusters. Therefore, the maximum number of edges of this type that can be used in a tour is 3. The second least costly type of edge is $B$ with weights of 3. This implies that no tour can be shorter than $3 \cdot 1 + 3 \cdot 3 = 12$. Each solution with a permutation in consecutive or reverse-consecutive order uses exactly three edges of weight $1$ and three edges of weight $3$ which implies a cost of $12$ and is therefore optimal. □

**Theorem 5.7.** *Starting with the solution consisting of all the white nodes and the permutation* $\pi = (1, 4, 5, 2, 3, 6)$*, CBLS is not able to achieve any improvement.*

*Proof.* Starting with all white nodes and a permutation $\pi = (1, 4, 5, 2, 3, 6)$, the solution contains three Type-$A$ edges of cost $1$ and three Type-$C$ edges of cost $4$. This implies a total cost of $15$ which is not optimal. The edges belonging to this tour are marked solid in Figure 5.5. We claim that this solution is locally optimal, i.e. can not be improved by a 2-opt step.

When a 2-opt move is performed, depending on the different types of edges that are removed from the current tour, we show that the resulting tours have costs greater than $15$.

Note, that all $3$ edges of cost $1$ are already used in the current permutation which implies no additional edge of cost $1$ can be added. We inspect the different 2-opt steps with respect to the edges that are removed.

- If two edges of type $A$ which have cost of 1 are removed, two other edges need to be added and the least costly edges that can be added have a weight of 3. This makes the total cost of the resulting solution to be at least $15 - 2 \cdot 1 + 2 \cdot 3 = 19$ which is greater than 15.

- If one edge of type $A$ (weight 1) and one edge of type $C$ (weight 4) are removed, again with the minimum two edges of cost 3 that are added, the total cost is at least $15 - 1 - 4 + 2 \cdot 3 = 16$ which is greater than 15.
- For removing two edges of Type $C$, there are three options:
  - Remove the edge between cluster 1 and cluster 4 and also the edge between cluster 2 and cluster 5. This 2-opt results in permutation $\pi' = (1, 5, 4, 2, 3, 6)$ which adds two edges of type $D$ to the solution, making the total cost greater than 15.
  - Remove the edge between cluster 1 and cluster 4 and also the edge between cluster 3 and cluster 6. This 2-opt results in permutation $\pi' = (1, 3, 2, 5, 4, 6)$ which also adds two edges of type $D$ to the solution, making the total cost greater than 15.
  - Remove the edge between cluster 2 and cluster 5 and also the edge between cluster 3 and cluster 6. This 2-opt results in permutation $\pi' = (1, 4, 5, 3, 2, 6)$ which also adds two edges of type $D$ to the solution, making the total cost greater than 15.

We have shown that no 2-opt step is accepted, which completes the proof. □

In contrast to the negative result for CBLS, we show that NEN-LS is able to reach an optimal solution when starting with the same solution.

**Theorem 5.8.** *Starting with $\pi = (1, 4, 5, 2, 3, 6)$, NEN-LS finds an optimal solution for the instance $G_1$ in $O(nm^2)$ steps.*

*Proof.* Starting with a solution with only white nodes and the permutation of $\pi = (1, 4, 5, 2, 3, 6)$, the lower level is already locally optimal using the arguments in the proof of Theorem 5.7. This implies that the solution does not change unless a grey note in cluster $V_1$ is selected.

Let $P = \{p_1, \cdots, p_6\}$ be the current set of spanning nodes. Selecting a grey node $p'_1$ for cluster $V_1$ leads to the set of spanning nodes $P' = \{p'_1, p_2, \cdots, p_6\}$. $P'$ in combination with the the current permutation $\pi = (1, 4, 5, 2, 3, 6)$ has a total cost of 111 as there is one edge of type $D$ with cost 100. We now show that starting from this solution and performing a 2-opt local search on the lower level results in an optimal solution.

In order to accept a new permutation on the lower level a solution of cost at most 111 has to be obtained. We do a case distinction according to the different types of edges that are removed in a 2-opt operation. If we only remove edges of type $A$ and $C$, we

reach a solution with total cost of greater than $111$ using the arguments in the proof of Theorem 5.7. Hence, we only need to consider the case where at least one edge of type $D$ is removed.

- There are two possibilities of removing one edge of type $D$ and one of the edge of type $C$ leading to the permutations $\pi' = (1, 5, 4, 2, 3, 6)$ and $\pi'' = (1, 3, 2, 5, 4, 6)$. Both have two edges of type $D$ which implies a total cost of greater than $111$ and are therefore rejected.
- Considering the case of removing the edge of type $D$ and one of the edges of type $A$, the only applicable 2-opt move leading to a different permutation results in the permutation $\pi' = (1, 2, 5, 4, 3, 6)$. The resulting solution has cost $16$ and is therefore accepted.

Considering $\pi' = (1, 2, 5, 4, 3, 6)$, the only acceptable 2-opt move leads to the global optimum $\pi_{opt} = (1, 2, 3, 4, 5, 6)$. The runtime is bounded by $O(nm^2)$ as it takes $O(n)$ time on the upper level to selected a grey node. Furthermore, each lower level optimization is bounded by $O(m^2)$ as either permutations are locally optimal with respect to the spanning nodes or there are at most two improvements of the permutation in the case that a grey node of cluster $V_1$ is selected. □

## 5.5 Benefits of Variable Neighbourhood Local Search

In this section we introduce an instance of the problem for which both of the mentioned neighbourhood search algorithms fail to find the optimal solution. Nevertheless, the combination of these approaches as described in Algorithm 12 results in finding the global optimum.

We consider the undirected complete graph $G_3$ shown in Figure 5.6 which has $6$ clusters each containing $n/6$ nodes. There are three kinds of nodes in this graph: white, grey and black. The first cluster consists of $n/12$ black, $n/24$ white, and $n/24$ grey nodes. All other clusters contain $n/12$ white and $n/12$ black nodes. We refer to the set of white, black and grey nodes of cluster $V_i$ by $V_{iW}$, $V_{iB}$, and$V_{iG}$, respectively.

There are 5 types of edges in this graph, 4 of which are quite similar to the 4 types of the instance in Section 5.3. The other type, named type $D$ below, includes the edges between two consecutive black nodes with a cost of $1.5$.

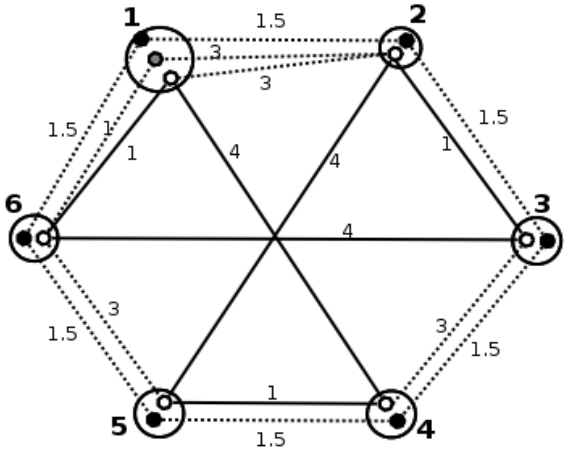


FIGURE 5.6: Graph $G_3$ showing one node of each type for each cluster and omitting edges of cost 100.

- Type $A$: Edges of this type have a cost of 1.

$$A = \{\{v_i, v_j\} \mid \quad (v_i \in V_{1W} \cup V_{1G} \wedge v_j \in V_{6W}) \ \vee \ (v_i \in V_{2W} \wedge v_j \in V_{3W}) \ \vee \ (v_i \in V_{4W} \wedge v_j \in V_{5W})\}$$

- Type $B$: Edges of this type have a cost of 3.

$$B = \{\{v_i, v_j\} \mid \quad (v_i \in V_{1W} \cup V_{1G} \wedge v_j \in V_{2W}) \ \vee \ (v_i \in V_{3W} \wedge v_j \in V_{4W}) \ \vee \ (v_i \in V_{5W} \wedge v_j \in V_{6W})\}$$

- Type $C$: Edges of this type have a cost of 4.

$$C = \{\{v_i, v_j\} \mid \quad (v_i \in V_{1W} \wedge v_j \in V_{4W}) \ \vee \ (v_i \in V_{2W} \wedge v_j \in V_{5W}) \ \vee \ (v_i \in V_{3W} \wedge v_j \in V_{6W})\}$$

- Type $D$: Edges of this type have a cost of 1.5.

$$D = \{\{v_i, v_j\} \mid (v_i \in V_{kB} \wedge v_j \in V_{(k+1)B} \ , \ 1 \leq k \leq 5) \ \vee \ (v_i \in V_{6B} \wedge v_j \in V_{1B})\}$$

- Type $F$: Edges of this type have a large cost of 100. All edges other than those of type $A$ or $B$ or $C$ or $D$ in this complete graph are of Type $F$. Note that the edges between grey nodes of the first cluster and the white nodes of the forth cluster are also of this type.

$$F = \quad E \setminus \{A \cup B \cup C \cup D\}$$

We now show that an optimal solution visits a black node from each cluster in consecutive or reverse-consecutive order. Then in Theorem 5.10, we show that the algorithms CBLS and NEN-LS may get stuck in local optimums.

*Property* 5.9. The optimal solution for the graph $G_3$ is visiting all black nodes with the consecutive or reverse-consecutive order.

*Proof.* There are three kinds of nodes in this graph; white, grey and black. Any solution that contains black and one other kind of node has at least two edges of type $F$ and weight 100 which makes the total cost of that solution more than 200. A solution that visits all black nodes in consecutive or reverse-consecutive order has 6 edges of type $D$ and a total cost of 9. On the other hand, if we consider only white and grey nodes, our graph is the same as the instance of Section 5.3 with the optimal solution of cost 12. Therefore, visiting all black nodes with the cost of 9 is the optimal solution. □

**Theorem 5.10.** *Starting with a spanning node set $P$ consisting of only white nodes and the permutation $\pi = (1, 4, 5, 2, 3, 6)$, CBLS and NEN-LS get stuck in a local optimum of $G_3$.*

*Proof.* We first show that the mentioned initial solution is a local optimum for CBLS. The cost of this solution is 15 which is less than any of the edges between black nodes and white or grey nodes which are of type $F$. Therefore, any solution consisting of two kinds of nodes, black and another kind, cannot be accepted after this solution. Considering only white and grey nodes, the permutation $\pi' = (1, 2, 3, 4, 5, 6)$ is better than the current one, but as we saw in Theorem 5.7 of Section 5.3 this order can not be achieved with Algorithm 8. A solution consisting of all the black nodes is less costly only if they are visited in the optimal order of $\pi' = (1, 2, 3, 4, 5, 6)$ which is exactly the same permutation that is better for white nodes as well. As we discussed, this permutation is not achievable by searching the 2-opt neighbourhood of the current solution and the Cluster-Based approach can not find it.

Now we investigate the behaviour of NEN-LS which performs a local search based on the Node-Based approach for this instance. We show that this algorithm finds another locally optimal solution. Starting with the initial solution that is specified in the theorem, all black nodes can not be selected in one step and trying any one of the black nodes is rejected, because using two edges of type $F$ are inevitable which makes the solution worse than the initial solution. The only spanning node set left in the NEN has the grey node of the first cluster. For this selection of nodes, the 2-opt TSP solver of the lower layer finds the optimal order of clusters similar to what we described in Theorem 5.8 of Section 5.3 which form a solution of cost 12. From this point any Node-Exchange-Neighbourhood search fails to find a better solution. □

Using the combination of the two hierarchical approaches by variable-neighbourhood search allows us to escape these local optima. As a result VNS obtains an optimal solution when starting with the same solution as investigated in Theorem 5.10.

**Theorem 5.11.** *Starting with a spanning node set $P$ consisting only of white nodes and the $\pi = (1, 4, 5, 2, 3, 6)$, VNS obtains an optimal solution in time $O(n^3)$.*

*Proof.* This approach is supposed to start with Cluster-Based algorithm and alternate between the two algorithms whenever CBLS is stuck in a locally optimal solution. As we saw, from the initial solution, Algorithm 8 can not find any better solutions, because the initial solution is a local optimum for that algorithm. Finding this out requires searching all the 2-opt neighbourhood which can be done in constant time, because the number of clusters is fixed. Then NEN-LS manages to find another solution with the permutation of $\pi' = (1, 2, 3, 4, 5, 6)$. This can also be done in polynomial time as we described in Theorem 5.8 of Section 5.3. Then CBLS uses this as a starting solution. As $\pi' = (1, 2, 3, 4, 5, 6)$ is an optimal permutation the optimal set of nodes $P$ consisting of all black nodes is found in time $O(n^3)$ on the lower layer. □

The investigations of this section have pointed out where a combination of the two hierarchical approaches into variable neighbourhood search gives a clear benefit to the optimization process as it is crucial for escaping local optima of the two single approaches.

## 5.6 Conclusion

Local search approaches have been shown to be very successful for solving the generalized travelling salesman problem. Two hierarchical local search approaches have been introduced in [66] for solving this problem, which we have investigated in this chapter from a theoretical perspective. The two approaches search different neighbourhoods in each layer. By presenting instances where they mutually outperform each other, we have gained new insights into the complimentary abilities of the two approaches. We have proved that there are instances that can be solved in polynomial time with one approach, while the other approach fails to find an optimal solution.

Furthermore, Hu and Raidl [66] have introduced a combination of the two approaches into a variable neighbourhood search algorithm. They have claimed that this algorithm, which searches both neighbourhoods, performs better for solving different instances of the GTSP. Supporting their idea, we have presented and analysed a class of instances

where combining the two approaches into a variable-neighbourhood search helps to escape from local optima of the single approaches.

# Chapter 6

# Simple Evolutionary Algorithms and the Generalized Travelling Salesman Problem

## 6.1 Introduction

In this chapter, we investigate the behaviour of simple evolutionary algorithms solving the generalized travelling salesman problem in the context of parameterized analysis. Similar to the previous chapter, we concentrate on Hu and Raidl's [66] two hierarchical approaches for GTSP which is defined in Section 3.4.1. We analyse a (1+1) EA based on the Cluster-Based approach and a (1+1) EA based on the Node-Based approach by presenting upper and lower bounds for optimization time. We show that the Cluster-Based approach gives us a Fixed-Parameter evolutionary algorithm while this is not the case for the Node-Based approach. However, we also show that the worst case instance presented for the Cluster-Based approach, can be solved in polynomial time by means of the Node-Based approach; hence, there are instances of the problem which the latter approach can solve more efficiently.

For finding the lower bound on the optimization time of the Node-Based (1+1) EA, we introduce a Euclidean class of instances which requires exponential time with respect to the number of clusters. To our knowledge currently an exponential lower bound for solving TSP by a stochastic search algorithm is available only for ant colony optimization in the non-Euclidean case [75]. Our instance for the GTSP places nodes on two different circles with radius $r$ and $r'$ of a given centre. Exploiting the geometric properties of this instance class, we show by Multiplicative Drift Analysis (Theorem 4.6) that

the evolutionary algorithm under investigation ends up in a local optimum which has different chosen nodes for almost all clusters. Leaving such a local optimum requires exponential time for many mutation-based evolutionary algorithms and leads to an exponential lower bound with respect to the number of clusters for the investigated algorithm.

The work of this chapter is based on an ECJ paper [19] and another paper that is submitted to a journal [102]. The outline of this chapter is as follows. Section 9.2 introduces the algorithms that are subject to our investigations in this chapter. The runtime analysis for the Cluster-Based (1+1) EA, which includes introducing a hard instance for this algorithm, is presented in Section 6.3. Then in Section 6.4 we show that the introduced hard instance of the Cluster-Based (1+1) EA can be solved in polynomial time with the Node-Based (1+1) EA. Section 6.5 continues analysis on the Node-Based (1+1) EA by finding lower and upper bounds for the optimization time of this algorithm. Finally, we finish with some concluding remarks in Section 9.6.

## 6.2 Simple Evolutionary Algorithms for the Generalised Travelling Salesman Problem

This section contains the description of two simple evolutionary algorithms that are analysed in this chapter in the context of parameterized complexity. The algorithms, namely Cluster-Based (1+1) EA and Node-Based (1+1) EA, are based on two hierarchical approaches for GTSP introduced by Hu and Raidl [66] and presented in Section 3.4.1.

### 6.2.1 Cluster-Based (1+1) EA

Similar to the Cluster-Based local search algorithm of Section 5.2.1, the upper layer solution in the Cluster-Based (1+1) EA is a permutation of clusters, $\pi$, and the lower layer solution is a set of nodes $P = \{p_1, \ldots, p_m\}$ with $p_i \in V_i$ that minimises the cost of a tour that respects $\pi$. Given the restriction imposed by $\pi$, 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 presented in Section 5.2.1.1. Assuming that $p_{\pi_i}$ is the chosen node for cluster $V_{\pi_i}$, $1 \leq i \leq m$, similar to Section 3.4.1 and Section 5.2, the cost of the solution $S = (P, \pi)$ is

$$c(S) = c(p_{\pi_m}, p_{\pi_1}) + \sum_{i=1}^{m-1} c(p_{\pi_i}, p_{\pi_{i+1}}).$$

**Algorithm 13:** Cluster-Based (1+1) EA

Choose a random permutation $\pi$ of the $m$ given clusters;
Let $P = \{p_1, \ldots, p_m\}$ be the optimal set of nodes for $\pi$ and $S = (P, \pi)$ be the resulting solution;
**while** *termination condition not satisfied* **do**
    $\pi' = \pi$;
    **for** $i \in \{1, \cdots, K\}$, *where* $K \sim 1 + \mathrm{Pois}(1)$ **do**
        Choose two nodes in $\pi'$ uniformly at random;
        Perform the *Jump* with the chosen nodes on $\pi'$;
    Let $P' = \{p'_1, \ldots, p'_m\}$ be the optimal set of nodes for $\pi'$ and $S' = (P', \pi')$ be the resulting solution;
    **if** $c(S') < c(S)$ **then**
        $S = S'$;

Our proposed algorithm (Algorithm 13) starts with a random permutation of clusters. In each iteration, a new solution $\pi'$ of the upper layer 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 $1 + \mathrm{Pois}(1)$, where $\mathrm{Pois}(1)$ 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*.

### 6.2.2 Node-Based (1+1) EA

In Node-Based approach, selecting the spanning nodes is done in the upper layer and the corresponding shortest Hamiltonian cycle is found in the lower layer. The Node-Based (1+1) EA is presented in Algorithm 14. In contrast to Node-Based local search algorithm of Section 5.2.2, the upper layer uses the (1+1) EA to search for the best spanning set instead of a local search method; hence, more than one change on the spanning set is possible on the upper layer at each iteration of the algorithm. The condition for accepting the new solution is a strict improvement and the definition of cost of a solution $S$ is similar to that of Section 6.2.1.

We analyse this algorithm with respect to the (expected) number of iterations on the upper layer, until it has found an optimal solution and call this the (expected) optimization time of the algorithm. Note that the lower layer consists of an NP-hard problem; hence, when showing polynomial upper bounds, we only consider instances where the lower layer can be solved in polynomial time. For general case, there exist very effective solvers for TSP such as Concorde [3], that can be used in the lower layer. Note that the lower layer does not need to solve an NP-hard problem in the Cluster-Based

**Algorithm 14:** Node-Based (1+1) EA

Let $P = \{p_1, p_2, \ldots, p_m\}$, where $p_i \in V_i$ are chosen uniformly at random;
Let $\pi$ be the optimal permutation for $G[P]$ and $S = (P, \pi)$ be the resulting solution;
**while** *termination condition not satisfied* **do**
    $P' = P$;
    **for** $i \in \{1, \cdots, m\}$ **do**
        with probability $1/m$, sample $p'_i \sim \text{Unif}(V_i)$;
    Let $\pi'$ be the optimal permutation for $G[P']$ and $S' = (P', \pi')$ be the resulting solution;
    **if** $c(S') < c(S)$ **then**
        $S = S'$;

approach. Nevertheless, we prove that there are instances that can be solved in polynomial time with Node-Based (1+1) EA, while the Cluster-Based (1+1) EA [19] needs exponential time to find an optimal solution for them.

## 6.3 Analysis on Cluster-Based (1+1)EA

In this section we provide the analysis on the behaviour of the Cluster-Based (1+1) EA on the GTSP. Upper and lower bounds on the optimization time of the algorithm are presented in Sections 6.3.1 and 6.3.2.

### 6.3.1 Upper Bound for Optimization Time

Theorem 6.1 below, presents an upper bound for the expected optimization time of the Cluster-Based (1+1) EA solving GTSP.

**Theorem 6.1.** *The expected optimisation time of the Cluster-Based (1+1) EA is* $O(m!m^{2m})$.

*Proof.* We consider the probability of obtaining the optimal tour $\pi^*$ on the global graph $H$ from an arbitrary tour $\pi$. 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 $i$th operation jumps element $\pi^*_i$ in $\pi$ to position $i$. Since the probability of $\text{Pois}(1) + 1 = m$ is $1/(\text{e}(m-1)!)$, the probability of a specific sequence of $m$ *Jump* operations to occur is bounded below by

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

Therefore, the expected waiting time for such a mutation is

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

which proves the upper bound on the expected optimisation time. □

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

### 6.3.2 Lower Bound for Optimization Time

In this section we find a lower bound for the optimisation time of the proposed algorithm. Figure 6.1 illustrates an instance $G_G$ of GTSP, for which finding the optimal solution is difficult by means of the presented hierarchical evolutionary algorithm with Cluster-Based approach. In this graph, each cluster has two nodes. On the upper layer a tour for clusters is found by the EA and on the lower layer 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 cycle 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 and 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 for the ones presented in the figure which have weight $1/m$. An optimal solution of

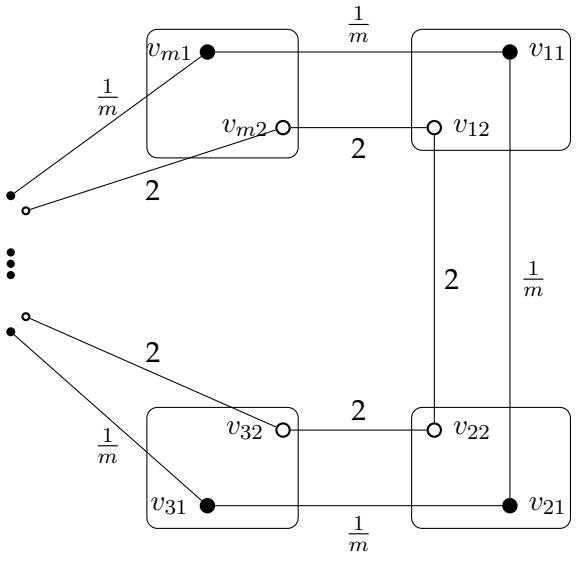


FIGURE 6.1: $G_G$, a hard instance of GTSP for Cluster-Based (1+1) EA.

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 6.1 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 edge with the global optimal solution.

The clusters are numbered in the figure, and a measure $S$ for evaluating cluster orders is based on this numbering: Let $\pi = (\pi_0, \ldots, \pi_{m-1})$ represent the permutation of clusters in the upper layer, then $S(\pi)$ defined below, indicates the similarity of $\pi$ with the optimal permutation. Observe that $S(\pi) \leq m$.

$$\begin{aligned} S(\pi) = |\{i \mid \quad & \pi_{(i+1 \mod m)} = \pi_i + 1 \ \wedge \ (\pi_i < m) \\ \vee \quad & (\pi_{(i+1 \mod m)} = 1) \ \wedge \ (\pi_i = m) \\ \vee \quad & (\pi_{(i+1 \mod m)} = \pi_i - 1) \ \wedge \ (\pi_i > 1) \\ \vee \quad & (\pi_{(i+1 \mod m)} = m) \ \wedge \ (\pi_i = 1)\}| \end{aligned}$$

A large value of $S(\pi)$ means that many clusters in $\pi$ are in the same order as in the optimal solution. Note that $S(\pi^*) = m$ for an optimal solution $\pi^*$. A solution $\pi$ 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 optimisation time.

**Lemma 6.2.** *Let $\pi$ and $\pi'$ 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 cycle 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 layer 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 layer 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 $a = S(\pi)$ in a solution $\pi$ be the number of clusters that their right neighbour is one of their adjacent clusters in the optimal solution. Then $b = m - a$ of clusters have a different neighbour on their right. If $\pi$ is not the optimal solution, then the lower layer 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 $\pi$ will be $c(\pi) = 2a+b = 2a+m-a = m+a$. Consider a solution $\pi'$ with $a' = S(\pi')$ and $S(\pi') > S(\pi)$. We have $c(\pi') = m + a' > m + a = c(\pi)$ which completes the proof. □

Lemma 6.2 shows that any non-optimal offspring $\pi'$ of a solution $\pi$ is not accepted if it is closer to an optimal solution $\pi^*$. 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 optimisation time as shown in the following theorem.

**Theorem 6.3.** *Starting with a permutation of clusters chosen uniformly at random, the optimisation time of the Cluster-Based (1+1) EA on $G_G$ is $(\frac{m}{2})^{\frac{2m}{3}}$ with probability $1 - \mathrm{e}^{-\Omega(m)}$.*

*Proof.* Considering $G_G$ illustrated in Figure 6.1, 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 $T = Cm^{3+\delta}$ steps where $C$ is an appropriate constant. For the typical run we show the following:

1. A local optimum $\pi$ 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 $\pi_{init}$, where $\varepsilon$ is a small positive constant.

We count the number of permutations in which at least $\varepsilon m$, $\varepsilon > 0$ a small constant, of cluster-neighbourhoods are correct. We should select $\varepsilon m$ of the clusters to be followed by their specific neighbour, and consider the number of different permutations of $m - \varepsilon m$ clusters:

$$\binom{m}{\varepsilon m}(m - \varepsilon m)! \tag{6.1}$$

Some solutions are double-counted in this expression, so the actual number of different solutions with $S(\pi) \geq \varepsilon m$ is less than (6.1). Therefore, the probability of having more than $\varepsilon m$ 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 most $\varepsilon m$ correctly ordered clusters.

Now we analyse the expected time to reach a solution $\pi$ with $S(\pi) = 0$. For this purpose, we first consider the *exchange* operation and find the minimum number of different exchanges at each step, that reduce the number of good orderings in a solution.

If we show the permutation of clusters for the current solution by $\pi = (\pi_1, ..., \pi_m)$, then there are $l = S(\pi)$ clusters in this permutation that are followed by their consecutive cluster. Note that for any solution other than the local optimum, $l > 0$ holds. Let $j$ be one of these clusters which is followed by cluster $j + 1$. In order to destroy this good ordering, cluster $j$ should be exchanged with a cluster $r$ which fulfils the following requirements:

1. cluster $r$ can not be a consecutive cluster of $j$'s current neighbours, i.e. if we name $j$'s neighbours $i$ and $k$, then $r$ can not be $i - 1$, $i + 1$, $k - 1$, $k + 1$ because these nodes will introduce a new good ordering to the solution if replace $j$.
2. the current position of cluster $r'$ in the permutation should not be before or after clusters $j - 1$ or $j + 1$, because replacing $r$ with $j$ would introduce new good orderings in that case.

Therefore, the total number of positions in the permutation which should not be selected as $r$ is at most $8$, meaning that there are $m-8$ choices for $r$ that result in reducing the number of good orderings. Since there are $l$ choices for $j$ and $m - 8$ choices for $r$, the total number of possible exchange operations to reach a permutation $\pi'$ with $S(\pi') < S(\pi)$ is:

$$l \cdot (m - 8)$$

On the other hand, it is possible to simulate each *exchange* operation with two *jump*s. For any $j$ and $r$, $exchange(j, r)$ can be implemented by performing $jump(j, r)$ and $jump(r, j + 1)$. The first jump will place $j$ before $r$, and the second one will place $r$ before $j + 1$. Now we find the probability that two jumps happen at one step and simulate one of the possible exchange operations as:

$$P = l \cdot (m - 8) \cdot \frac{1}{e} \cdot \frac{1}{m^2} \cdot \frac{1}{m^2}$$

In the above formula, $l \cdot (m - 8)$ is the number of different choices for exchange operation, $\frac{1}{e}$ is the probability of performing two mutation operations at one step and each

$\frac{1}{m^2}$ is the probability of selecting the two specific nodes for a jump operation. Using this probability, the expected time until $l$ decrease by one is

$$E(T) = \frac{1}{P} = \gamma \cdot \frac{m^3}{l}$$

where $\gamma$ is an appropriate constant. The maximum value of $l$ is $S(\pi_{init})$ which we have already proved that is at most $\epsilon m$. Therefore, with summing up the expected time of reducing $l$ gradually from its maximum value to 1, we can find the expected time to reach the local optimum $\pi$ with $S(\pi) = 0$ as

$$E(T_{LO}) = \sum_{l=\epsilon m}^{1} \gamma \cdot \frac{m^3}{l} = O(m^3 \log m)$$

If $\gamma'$ denotes the appropriate constant that $E(T_{LO}) = \gamma' \cdot m^3 \log m$ then by Markov's inequality we have:

$$Pr\left(T_{LO} > 2 \cdot \gamma' \cdot m^3 \log m\right) \leq \frac{1}{2}$$

If we repeat phases of $2 \cdot \gamma' \cdot m^3 \log m$ iterations for $\frac{m^\delta}{\log m}$ times, the probability that the local optimum is not reached in any of them is at most:

$$Pr\left(T_{LO} > 2 \cdot \gamma' \cdot m^3 \log m \cdot \frac{m^\delta}{\log m}\right) \leq (\frac{1}{2})^{\frac{m^\delta}{\log m}} = e^{-\Omega(m^\delta)}$$

As a result, with probability $1 - e^{-\Omega(m^\delta)}$ the algorithm will reach a local optimum in a phase of $2\gamma' \cdot m^{3+\delta}$ steps which, if we consider $C = 2\gamma'$, is actually the same as the phase of $T = C \cdot m^{3+\delta}$ iterations that we mentioned previously.

To prove that with high probability, the global optimum is not reached during the considered phase, first note that by Lemma 6.2, 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 $\pi$ 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 $\pi$ with $S(\pi) \leq \varepsilon m$ in a single mutation step. We now upper bound the probability of such a direct jump which changes at least $(1-\varepsilon)m$ clusters to their correct order. Such a move needs $k \geq \frac{(1-\varepsilon)m}{3}$ operations in the same iteration because each *Jump* can change at most 3 edges. Taking into account that these *Jump* operations may be acceptable in any order, the probability of such a direct jump is at most

$$\frac{1}{e(k)!} \cdot \frac{1}{m^{2(k)}} \cdot (k)! = m^{-\Omega(k)} = m^{-\Omega(m)}. \tag{6.2}$$

So in a phase of $O(m^{3+\delta})$ iterations the probability of having such a direct jump is by union bound at most $m^{-\Omega(m)+3+\delta} = m^{-\Omega(m)}$.

So far we have shown that a local optimum $\pi$ with $S(\pi) = 0$ is reached with probability $1 - \mathrm{e}^{-\Omega(m^\delta)}$ within the first $T = Cm^{3+\delta}$ iterations. The probability of obtaining an optimal solution from a solution $\pi$ with $S(\pi) = 0$ is at most

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

We now consider an additional phase of $(\frac{m}{2})^{\frac{2m}{3}}$ steps after having obtained a local optimum. Using the union bound, the probability of reaching the global optimum in this phase is at most

$$\left(\frac{m}{2}\right)^{\frac{2m}{3}} \cdot \mathrm{e}^{-1} \cdot m^{-\frac{2m}{3}} \leq \left(\frac{1}{2}\right)^{\frac{2m}{3}}.$$

As a result, the probability of not reaching the optimal solution in these $(\frac{m}{2})^{\frac{2m}{3}}$ iterations is $1 - 2^{-\frac{2m}{3}} = 1 - \mathrm{e}^{-\Omega(m)}$. Altogether, the optimisation time is at least $(\frac{m}{2})^{\frac{2m}{3}}$ with probability $1 - \mathrm{e}^{-\Omega(m)}$. □

## 6.4 Benefit of Nodes-Based (1+1)EA

In this section, we show that the hard instance for Cluster-Based (1+1) EA introduced in [19] can be solved in polynomial time by the Node-Based approach. In order to do so, we first analyse how an optimal TSP tour can be found for this instance on the lower layer of the Node-Based approach in Section 6.4.1. Then in Section 6.4.2 we analyse the behaviour of Node-Based (1+1) EA on $G_G$.

### 6.4.1 Finding the Optimal Lower Layer Solution for $G_G$

The lower layer of the Node-Based approach needs to find an optimal TSP solution with respect to the node selection that has been done on the upper layer. Although solving TSP in general is NP-hard, it can be solved in polynomial time for the instances induced by picking one node of each cluster of the graph $G_G$. Algorithm 15 provides such a method. In step 4 of this algorithm, if the number of white nodes is at most 3, finding the shortest path can be done by checking all configurations. If the number of white nodes is more than 3, only edges of cost 1 will be used in the shortest path,

**Algorithm 15:** A Lower Layer TSP Solver

1: Consider $G[P]$, the graph induced by the given spanning set $P$
2: Consider $G'$ a graph consisting of nodes of $G[P]$ with no edges
3: Add the edges of $G[P]$ that have a cost of $1/m$ to $G'$
4: Find a shortest path visiting all white nodes in $G[P]$ and add the edges of that to $G'$.
5: Use edges of cost $m^2$ to make a Hamiltonian cycle out of the paths that are formed in $G'$ and the disconnected black nodes

and finding it can be done by a depth-first-search and checking all configurations of connecting the last 4 nodes of the path. Therefore step 4 needs time $O(m)$ to find the shortest path on white nodes. Since the required time for other steps of the algorithm is also at most $O(m)$, we can conclude that Algorithm 15 runs in time $O(m)$.

To prove that Algorithm 15 finds the optimal tour with respect to the spanning set fixed on the upper layer, we first present a property on the solutions of lower layer. Then in Lemma 6.5 we show that Algorithm 15 finds the optimal tour.

*Property* 6.4. Let $C(S)$ denote the total cost of a solution $S$. Also, let $Y$ and $X$ be two solutions with $r$ and $s$ edges of weight $m^2$ respectively. If $r > s$ then we have $C(Y) > C(X)$.

*Proof.* For a solution $Y$ with $r$ edges of weight $m^2$, the minimum total cost $C_{\min}(Y)$ is at least $r \cdot m^2$. On the other hand, the cost of the solution $X$ with $s$ edges of weight $m^2$, is maximised if all other $m - s$ edges have a cost of 2, as 2 is the second largest weight of the input. Therefore, the maximum cost $C_{\max}(X)$ of solution $X$ is at most $s \cdot m^2 + 2(m - s)$ and we have $C(X) \leq C_{\max}(X) \leq s \cdot m^2 + 2(m - s)$.

Since $r > s$, we also have: $s \cdot m^2 + 2(m - s) < r \cdot m^2 \leq C_{\min}(Y) \leq C(Y)$. Therefore, $C(X) < C(Y)$. □

**Lemma 6.5.** *Let $w \geq 4$ and $r = m - w$ be the number of white and black nodes selected on the upper layer, respectively. Moreover, let $s$ be the number of black nodes where the selected node in proceeding cluster with respect to the optimal solution is also black. Algorithm 15 finds an optimal tour with total cost of $s \cdot \frac{1}{m} + (m - r - 1) + (r - s + 1) \cdot m^2$.*

*Proof.* Consider a black node, for which one (or two) of the edges of cost $1/m$ is possible to be selected. If the lower layer does not select that edge, then it must select some other edge connected to that black node, to arrange a complete Hamiltonian cycle. Since all other edges of black nodes have a cost of $m^2$, refusing to select any one of $1/m$-edges results in increasing the number of $m^2$-edges by at least one. By Property 6.4, we know that solutions with greater number of $m^2$-edges are more costly; therefore, selecting $s$

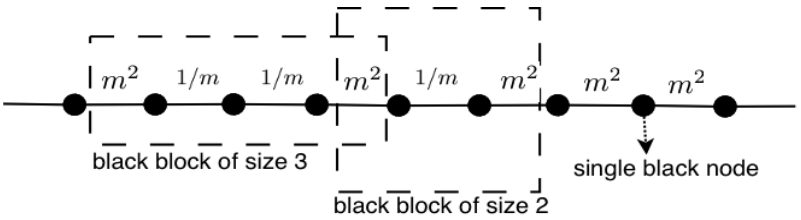


FIGURE 6.2: Blocks of black nodes

edges of kind $1/m$ on the lower layer is a must. This explains step 3 of the algorithm which forms a number of chains consisting of black nodes and $1/m$-edges.

On the other hand, in order to minimise the number of white-black connections which are of cost $m^2$, all white nodes need to form one chain which is done is step 4 of the algorithm, using edges of cost 1. This chain will be connected to two black nodes from its two ends.

So far, we have formed some chains of black nodes and one chain of white nodes. In order to connect these chains together, we can only use edges of weight $m^2$, because we have already used all $\frac{1}{m}$ edges and all other connections to black nodes have a cost of $m^2$. The number of these edges, which are added in step 5 of the algorithm, is $r-s+1$.

Overall, the optimal tour on the selected set of nodes consists of $s$ edges of weight $\frac{1}{m}$, $r-s+1$ edges of weight $m^2$ and $m-s-(r-s+1)$ edges of weight 1. Summing up the weight of the edges, we can find the total cost of the tour as stated in the lemma. □

### 6.4.2 Behaviour of Node-Based (1+1) EA on $G_G$

In the analysis of this section, we only consider the number of steps that the upper layer needs. Note that the lower layer uses Algorithm 15 which adds only a factor of $O(m)$ to our analysis. We start this section with a definition that helps us in describing a TSP tour that the lower layer forms. Then we present the lemmata which help us prove that with high probability the Node-Based (1+1) EA finds the optimal solution of $G_G$ in time $O(m^2)$. In the following, $w$ denotes the number of white nodes in the solution. Lemmata 6.9 and 6.11 show that the optimal solution will be reached in time $O(m^2)$ if we don't face a situation where $w \leq 3$, while Lemma 6.12 investigates the behaviour of the algorithm if we face that situation.

**Definition 6.6.** A **black block of size** $l$, $l > 0$ an integer, is a path on exactly $l$ consecutive black nodes, which consists of $l-1$ edges of cost $1/m$.

The two end nodes of a black block are connected to edges of cost $m^2$. Black blocks of size 1, 2 and 3 nodes are illustrated in Figure 6.2. If all the black nodes of a block mutate

to white, in the new solution that Algorithm 15 finds, there would be $l$ more nodes in the chain of white nodes, $l-1$ less edges of cost $\frac{1}{m}$ and one less edge of cost $m^2$. As a result, the total tour cost decreases by $m^2 + \frac{l-1}{m} - l$. Note that, in order to improve the cost, all these black nodes have to mutate in one step; otherwise, two edges of cost $m^2$ will remain in the sub-tour and the total cost increases.

**Lemma 6.7.** *If $w \geq 4$, then $w$ can only increase in the same steps that the number of $m^2$-edges decreases.*

*Proof.* From Lemma 6.5 we know that the total cost of a solution that has more than 3 white nodes is $s \cdot \frac{1}{m} + (m - r - 1) + (r - s + 1) \cdot m^2$, in which $r$ is the number of black nodes, $s$ is the number of edges of weight $\frac{1}{m}$, $m - r - 1$ is the number of edges that connect white nodes, and $(r - s + 1)$ is the number of $m^2$-edges. Therefore, if the number of white nodes increases, only a decrease in the number of $m^2$-edges can prevent an increase in the total cost. As a result, the number of white nodes can only increase in the same steps that the number of $m^2$-edges decrease. □

**Lemma 6.8.** *During a phase of $O(m^2)$ steps, if we do not face a situation where $w \leq 3$, with probability $1 - o(1)$, the sum of all increments on the number of white nodes is at most $5m$.*

*Proof.* Since $w > 3$, from Lemma 6.7 we know that the number of white nodes can increase only when the number of blocks reduces. The probability that a block of size at least 4 mutates to white in one step is at most $\frac{1}{(2m)^4}$. There are at most $\frac{m}{6}$ block of this size or more in a solution; therefore, the probability that one of them mutates to white at one step is at most $\frac{1}{12(2m)^3}$. Hence, the probability that one of them mutates in a phase of $C \cdot m^2$ steps is $O(\frac{1}{m})$. Therefore, with probability at least $1 - o(1)$, no black block of size 4 or more mutates to white. In other words, all blocks that mutate to white in a phase of $C \cdot m^2$ steps are of size at most 3. The number of blocks can reduce at most $m$ times, implying at most $3m$ mutations from black to white.

However, at each step that the number of blocks is reduced, some additional nodes, other than the nodes of that block, may also mutate to white. Let $X_{ij}$ be a random variable such that $X_{ij} = 1$ if node $j$ is selected for mutation at step $i$. The expected value of $X = \sum_{i=1}^{m} \sum_{j=1}^{m} X_{ij}$ is $E[X] = \sum_{i=1}^{m} \sum_{j=1}^{m} \frac{1}{m} = m$ and by Chernoff bounds we get $\text{Prob}(X \geq 2m) \leq e^{-\Omega(m)}$. Therefore, with probability $1 - e^{-\Omega(m)}$ at most $2m$ additional nodes mutate during the steps at which a black block is changed to white. As a result, at most $3m + 2m$ black nodes mutate to white in a phase of $m^2$ steps. □

**Lemma 6.9.** *If we do not face a situation where $w \leq 3$ or a situation with no black nodes, then with probability $1 - o(1)$, the optimal solution is reached in time $c \cdot m^2$, where $c = 24e$.*

*Proof.* Since the number of black nodes is at least one, there is always at least one white node that, if it mutates to black, increases the length of a black block. This move is accepted by the algorithm, because it shortens the white path by eliminating an edge of cost 1, while adds one edge of cost $\frac{1}{m}$ to the black block. The probability of this move is at least $\frac{1}{2\cdot m}\cdot\left(1-\frac{1}{m}\right)^{m-1}\geq\frac{1}{2\cdot m}\cdot\frac{1}{e}$, where $\frac{1}{2\cdot m}$ is the probability of deciding to mutate the proper white node and also to select the black node from its cluster. Moreover, $\left(1-\frac{1}{m}\right)^{m-1}$ is the probability that no other mutations happen at the same step for the other nodes.

Since the number of white nodes in the initial solution is at most $m$, and with probability $1-o(1)$ during $m^2$ steps at most $5m$ black nodes turn into white (Lemma 6.8), at most $6m$ steps of increasing the number of black nodes is sufficient for reaching the optimal solution. Let $X=\sum_{i=1}^{T}X_i$, where $X_i$ is a random variable such that $X_i=1$ if a white node mutates to black at step $i$, and $X_i=0$ otherwise. At each step $i$, before reaching the optimal solution, $\text{Prob}(X_i=1)\geq\frac{1}{2em}$. Considering a phase of $T=24em^2$ steps, by linearity of expectation we get $E[X]\geq 24em^2\cdot\frac{1}{2em}=12m$. Using Chernoff Bounds we get $\text{Prob}\left(X\leq(1-\frac{1}{2})12m\right)\leq e^{-\Omega(m)}$. As a result, with probability $1-e^{-\Omega(m)}$, $6m$ white nodes mutate to black which results in reaching the optimal solution in a phase of $24em^2$ steps. Overall with probability $1-o(1)$, the optimal solution is reached in time $24em^2$. □

**Lemma 6.10.** *The initial solution chosen uniformly at random, has at least $\frac{m}{48}$ single black nodes with probability $1-e^{-\Omega(m)}$*

*Proof.* Considering the consecutive clusters with respect to their optimal permutation, for any specific cluster, a black (or white) node may be selected for its following cluster with a probability of $1/2$. As a result, any selection of nodes in 3 consecutive clusters can happen with probability $(1/2)^3$. There are at least $m/3$ separate sets of consecutive clusters; therefore, the expected number of single black nodes is at least $\frac{m}{3\cdot 8}$. Using Chernoff bounds and considering $X$ to be the number of single black nodes in the initial solution, we have: $P(X<(1-1/2)\frac{m}{3\cdot 8})\leq e^{-\frac{m}{3\cdot 8}\cdot\frac{1}{8}}$

As a result, with a probability $1-e^{-\Omega(m)}$ the initial solution has at least $\frac{m}{48}$ single black nodes as described. □

In the proof of next lemma, we use the Simplified Drift Theorem presented in Theorem 4.7. In this theorem, as described in Section 4.5.3, $F_t$ denotes a filtration on states. In the proof of Lemma 6.11, we analyse the changes on the size of one large black block, and the filtration is done according to the steps where an accepted change happens on the size of that block.

**Lemma 6.11.** *With probability $1-o(1)$, the number of black nodes is at least one during $c \cdot m^2$ steps of the Node-Based (1+1) EA, where $c = 24e$.*

*Proof.* Let $r$ be the number of all black blocks in the solution. From Lemma 6.10 we know that with high probability, the initial solution consists of at least $\frac{m}{48}$ single black nodes. As a result, in the initial solution $r = \Omega(m)$.

In order to reach a solution in which all nodes are white, the number of black blocks need to reduce. Let's consider the step when for the first time $r \leq m^\epsilon$, where $\epsilon > 0$ is a small constant. At this step, $r \geq \frac{m^\epsilon}{2}$; otherwise, at least $\frac{m^\epsilon}{2}$ mutations have had happened at one step which is exponentially unlikely.

We first show that we either have a block of size greater than one at this stage, or we will reach such a situation. Let us assume that all of the blocks at this stage are of size one. For a single black node, there is a probability of $P_1^+ \geq \frac{2}{2 \cdot e \cdot m}$ that a white node mutates to black and extends the size of that block. On the other hand, the probability that this single black node mutates to white is $P_1^- \leq \frac{1}{2 \cdot m}$. Therefore, if a change happens on the size of this block, it would be an increase with probability at least

$$P_{1N}^+ = \frac{P_1^+}{P_1^+ + P_1^-} \geq \frac{\frac{1}{em}}{\frac{1}{em} + \frac{1}{2m}} \geq \frac{2}{2+e}$$

Therefore, the probability that none of these blocks experience an increase in the size when they change for the first time, is $\left(1 - \frac{2}{2+e}\right)^{\frac{m^\epsilon}{2}} = e^{-\Omega(m^\epsilon)}$. As a result, with probability $1 - e^{-\Omega(m^\epsilon)}$, we reach a stage (in a phase of at most $m^{1+\epsilon}$ steps) at which there are $r \leq m^\epsilon$ blocks and one of the blocks is of size at least 2. We refer to this block as the large block.

For a black block of size $l \geq 2$, there is a probability of $P_l^+ \geq \frac{2}{2 \cdot e \cdot m}$ that a white node mutates to black and extends the size of that block. But to decrease the size of the block, either the whole block needs to mutate at one step, or one improving move needs to happen somewhere else at the same step that a black node of the large block is mutating to white. An improving move can be a mutation on a white node that extends a black block, which happens with probability at most $\frac{2}{2m}$ for each block, or a mutation on all black nodes of a block, probability of which is upper bounded by $\frac{1}{2m}$ for each block. Since the number of blocks is at most $m^\epsilon$, the probability of an improving move to happen, is at most $\frac{2 \cdot m^\epsilon}{2m} + \frac{m^\epsilon}{2m}$. Overall, the probability of decreasing the size of the large block is

$$P_l^- \leq \frac{1}{(2m)^l} + \frac{2}{2m} \cdot \left(\frac{2 \cdot m^\epsilon}{2m} + \frac{m^\epsilon}{2m}\right) \leq \frac{1}{(2m)^l} + \frac{1}{m} \cdot \frac{3 \cdot m^\epsilon}{2m} \leq \frac{4m^\epsilon}{2m^2}$$

Now consider a phase of $m^{\frac{3}{2}}$ steps. With probability at most $\frac{4m^\epsilon}{2m^2} \cdot m^{\frac{3}{2}} = \frac{2m^\epsilon}{\sqrt{m}}$ the size of the large block is decreased at least once. Therefore, with probability $1 - O(m^{\epsilon-1/2}) = 1 - o(1)$ its size is not decreased in the mentioned phase.

On the other hand, there is a probability of at least $P_l^+ \geq \frac{1}{e \cdot m}$ at each step, that the size of the block is increased. Let $X_i$ be a random variable such that $X_i = 1$ if the size of large block is increased at step $i$, and $X_i = 0$ otherwise. The expected increase in the size of that block in a phase of $m^{\frac{3}{2}}$ steps is $\sum_{i=0}^{m^{3/2}} X_i \geq \frac{\sqrt{m}}{e}$. Moreover, by Chernoff bounds (Section 4.2.2) we have $\sum_{i=0}^{m^{3/2}} X_i \geq \frac{\sqrt{m}}{2e}$ with probability at least $1 - e^{-\Omega(\sqrt{m})}$.

Overall, with probability $1 - o(1)$, the size of the large block is at least $\frac{\sqrt{m}}{2e}$ after a phase of $m^{\frac{3}{2}}$ steps.

After this phase, we consider a phase of $c \cdot m^2$, $c = 24e$, steps and show that with high probability, the large black block does not lose more than half of its nodes. In order to show this, we use the Simplified Drift Theorem (Theorem 4.7). Let $t_0$ be the first step after the previous phase has finished and let $L$ be the largest block at that time. We define $X_t$, $t \geq 0$, as

$$\begin{aligned} X_t := \quad & \text{size of } L \text{ at } t_0 \\ + \quad & \text{the number of steps increasing size of } L \text{ from } t_0 \text{ until } t_0 + t \\ - \quad & \text{the number of nodes removed from } L \text{ from } t_0 \text{ until } t_0 + t. \end{aligned}$$

Note that $X_t$ always represents a lower bound on the size of $L$ at time $t + t_0$. We filter the steps and only consider the relevant steps, i. e. the steps in which a change happens on the size of $L$. Moreover, we set $a = \frac{X_0}{2}$, $b = X_0$, $r = 1$, $\varepsilon = \frac{1}{4e}$ and $\delta = 1$.

Earlier, we found an upper bound on $P_l^-$ and a lower bound on $P_l^+$. An upper bound on the latter is $P_l^+ \leq \frac{2}{2m}$, because in order to increase the size of a black block, at least one of the two white neighbours of it need to mutate to black. Using these bounds, we get upper and lower bounds for $P_{rel} = P_l^+ + P_l^-$, the probability of each step to be a relevant step:

$$\frac{1}{e \cdot m} \leq P_{rel} \leq \frac{1}{m} + \frac{4m^\epsilon}{2m^2} \leq \frac{2}{m}$$

At each step, with probability at least $\frac{1}{em}$, an increase happens on the size of $L$; hence, the positive drift on $X_t$ is $\frac{1}{em}$, and the positive drift on $X_t$ in the relevant steps is:

$$\Delta^+ \geq \frac{1}{em} \cdot \frac{1}{P_{rel}} \geq \frac{1}{em} \cdot \frac{m}{2} \geq \frac{1}{2e}$$

Also, the expected decrease in the number of black nodes of that block, in the relevant steps is

$$\Delta^- \leq \left(\frac{l}{(2m)^l} + \left(\sum_{k=1}^{m} k \cdot \frac{k+1}{(2m)^k}\right) \cdot \left(\frac{2 \cdot m^\epsilon}{2m} + \frac{m^\epsilon}{2m}\right)\right) \cdot \frac{1}{P_{rel}}$$

$$\leq \left(\frac{1}{(2m)^l} + \frac{2}{m} \cdot \frac{3 \cdot m^\epsilon}{2m}\right) \cdot \frac{1}{P_{rel}} \leq \frac{4m^\epsilon}{m^2} \cdot \frac{1}{P_{rel}} \leq \frac{4m^\epsilon}{m^2} \cdot em \leq \frac{4em^\epsilon}{m}$$

where $k$ is the number of black nodes that are removed from the large block, and $\frac{k+1}{(2m)^k}$ is the probability of such mutations to happen in one step. Here, $k+1$ is the number of possible ways that $L$ can lose $k$ nodes, and $\frac{1}{(2m)^k}$ is the probability that those nodes mutate to white. Moreover, $\sum_{k=1}^{m} k \cdot \frac{k+1}{(2m)^k} \leq \frac{2}{m}$ holds for $m \geq 3$. Using $\Delta^-$ we find the total expected difference of

$$E[X_{t+1} - X_t \mid F_t \wedge a < X_t < b] = \Delta^+ - \Delta^- \geq \frac{1}{2e} - \frac{4em^\epsilon}{m}$$

Therefore, the first condition of the simplified drift theorem holds. The second condition also holds because at each step $X_t$ can be increased by at most 1 and the probability of decreasing it by $j$ is

$$\text{Prob}(X_t - X_{t+1} \mid F_t \wedge a < X_t < b) \geq j) \leq \frac{j+1}{(2m)^j} \cdot \frac{4 \cdot m^\epsilon}{2m} \cdot \frac{1}{P_{rel}} \leq \frac{1}{2^j}.$$

Therefore, the conditions of simplified drift theorem hold and we get

$$\text{Prob}(T^* \leq 2^{c^* \cdot X_0/2}) = 2^{-\Omega(X_0/2)},$$

As a result, with probability $1 - 2^{-\Omega(\sqrt{m})}$, the size of the large block does not decrease to less than $a$, in a phase of $c \cdot m^2$ steps. Overall, with probability $1 - o(1)$, the number of black blocks is at least one during the mentioned phase. □

**Lemma 6.12.** *From the situation where $w \leq 3$, with probability $1 - o(1)$, the optimal solution is reached in time $O(m^{1+\epsilon})$, where $\epsilon > 0$ is a constant.*

*Proof.* In the situation where there are only $w \leq 3$ white nodes in the solution, there can be at most 3 blocks of black nodes. At most 2 of these blocks can contain only one node. We first show that all single black blocks (if any exist) either mutate to white or become larger, in a phase of $O(m^{1+\epsilon})$ steps, resulting in a solution with $w \leq 5$ and no single black block. Then we show that with high probability, from that solution the optimal solution is found in $O(m^{1+\epsilon})$.

While there exists a single black block, there exists a black to white mutation that decreases the number of blocks, and there exist 2 white nodes that can mutate to black

and extend the single black block. These moves happen with probability at least $\frac{3}{2 \cdot e \cdot m}$ at each step, and reduce the number of single black blocks by at least one. Therefore, in a phase of $\frac{4 \cdot e \cdot m}{3}$ steps, with probability at least $\frac{1}{2}$, at least one of the single black blocks is destroyed. As a result in a phase of $\frac{4 \cdot e \cdot m^{1+\epsilon}}{3}$ steps, with probability at least $1 - (\frac{1}{2})^{m^\epsilon}$, at least one single black block is destroyed. Therefore, in a phase of $\frac{8 \cdot e \cdot m^{1+\epsilon}}{3}$ steps, with probability $1 - e^{-\Omega(m^\epsilon)}$, all initial single black blocks are destroyed.

Now we need to show that in that phase, the algorithm does not produce any new single black blocks. To create a single black block, either a white node should mutate to black, or a black block of size $k$ should lose $k-1$ black nodes, $k \geq 2$. Both of these moves need to happen at the same step where another improving move has happened, which can be reducing the number of black blocks, or extending a black block by mutating a white node to black. There are at most 3 white nodes and at most 2 single black blocks in this situation. Therefore, the probability of an improving move at each step is at most $\frac{5}{2m} + \frac{3}{(2m)^2}$. As a result, the probability of producing a new single black block is $(\frac{3}{2m} + \frac{k}{(2m)^{k-1}})(\frac{5}{2m} + \frac{3}{(2m)^2})$, which implies that in a phase of $O(m^{1+\epsilon})$, this move does not happen with probability $1 - o(1)$.

Now we investigate the situation where $w \leq 5$, and there is no single black block. While there is at least one white node in the solution, there is a probability of at least $\frac{1}{2 \cdot e \cdot m}$ to decrease the number of white nodes at each step. This implies that in a phase of $20 \cdot e \cdot m^{1+\epsilon}$ steps, with probability $1 - e^{-\Omega(m^\epsilon)}$, at least 5 white nodes mutate to black. Now we show that with high probability, no black node is mutated to white in this phase which completes the proof.

In order to accept mutating a black node to white, either all black nodes of a block have to mutate at the same step, or a black node at one end of a block should mutate to white at the same step that another improving mutation has happened. At each step, the probability of the former is at most $\frac{3}{(2m)^2}$, because there are at most 3 blocks and we have assumed that the size of each of them is at least 2; and the probability of the latter is at most $\frac{6}{2m} \cdot \frac{5}{2m}$, because there are at most 6 black nodes at one end of a block, and 3 white nodes. In a phase of $20 \cdot e \cdot m^{1+\epsilon}$ steps, the probability that at least one of these happens is at most $O(m^{1-\epsilon})$. Therefore, with probability $1 - o(1)$, no black node is mutated to white in this phase. □

**Theorem 6.13.** *Starting from an initial solution chosen uniformly at random, the Node-Based (1+1) EA finds the optimal solution of $G_G$ in time $O(m^2)$ with probability $1 - o(1)$.*

*Proof.* Lemma 6.11 shows that in a phase of $c \cdot m^2$, $c = 24e$, steps, the number of black nodes doesn't decrease to 0 with probability $1 - o(1)$. Therefore, due to Lemma 6.9, if we never face a situation where $w \leq 3$, the optimal solution is found in time $O(m^2)$

with probability $1 - o(1)$. Moreover, if we face a situation where $w \leq 3$, according to the Lemma 6.12, with probability $1 - o(1)$ it takes $O(m^{1+\epsilon})$ additional steps to find the optimal solution where $\epsilon > 0$ is a constant. Overall, with probability $1 - o(1)$, in time $O(m^2)$, the optimal solution is found by the Node-Based (1+1) EA. □

Considering only 1-bit flips, we get the following result for NEN-LS*.

**Corollary 6.14.** *Starting with a solution chosen uniformly at random, NEN-LS* finds with probability* $1 - o(1)$ *the optimal solution of* $G_G$ *in time* $O(m^2)$.

## 6.5 Analysis on Nodes-Based (1+1)EA

In this section we provide the analysis on the behaviour of the Node-Based (1+1) EA on the GTSP. We present upper and lower bounds on the optimization time of the algorithm in Sections 6.5.1 and 6.5.2 respectively.

### 6.5.1 Upper Bound for Optimization Time

The lower layer in this approach is responsible for finding a Hamiltonian cycle which is an NP-hard problem and in the general case, takes exponential time to be solved. We here find an upper bound of the expected optimization time of the upper layer of our algorithm without taking into account the time needed for the lower layer. Then for the lower layer we restrict our analysis to the case where the problem is a Euclidean TSP with a number of inner points ($k$).

**Theorem 6.15.** *The expected optimization time of Node-Based (1+1) EA described in Algorithm 14 is* $O(n^m)$.

*Proof.* The proof of this theorem is to a great extent similar to the proof of Theorem 1 in [18] which finds an upper bound for the bi-level GMSTP with Node-Based Approach. Here we just describe the proof briefly.

Any arbitrary solution has at most $m$ clusters with a suboptimal selected node. The probability that the algorithm decides to change the selected node of those clusters in one step is at least $(1/m)^m$. Moreover, the probability that the new selected nodes in all clusters are the optimal nodes is

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

Therefore, the probability of reaching the optimal spanning set in the upper layer in one step is in $\Omega(n^{-m})$ and the expected time for that is in $O(n^m)$. □

Now we describe the case where the problem is a Euclidean TSP with $k$ inner points as studied by [88]. A Euclidean TSP instance can be partitioned into two sets of points: *outer points*, which are the vertices that comprise a convex hull of all the points, and *inner points*, which are the points that lie interior of the convex hull. Besides, since an optimal tour does not intersect itself, all outer points appear in the shortest Hamiltonian cycle in the same order as they appear in the boundary of convex hull [105]. This order can be found in $O(n \log n)$ [23] and can be merged with any fixed permutation of inner points in $O(kn)$ [28] to find the shortest Hamiltonian cycle with respect to the orders in both subsets, where $k$ is the number of inner points. If we try all the permutations of the inner points, then the overall optimization time of this layer will be in $O(k!kn)$. For large $k$ where it could take too much time to try all $k!$ permutations, one can use an evolutionary algorithm for finding the optimal permutation of inner points as well. For a (1+1) EA described in [88] and mutation operations of *jump* and *inversion*, it has been shown that an optimal solution is obtained in time $O(k!k^{2k})$ and $O((k-1)!k^{2k-2})$ respectively. Taking into account the time needed for merging inner and outer points, the expected optimization time of that algorithm for *jump* and *inversion* is $O(knk!k^{2k})$ and $O(kn(k-1)!k^{2k-2})$ respectively. With the upper bound for the expected time of finding the optimal spanned set in the upper layer that we found in Theorem 6.15, the overall expected time of our (1+1) EA would be $O(n^m knk!k^{2k})$ and $O(n^m kn(k-1)!k^{2k-2})$ for *jump* and *inversion* respectively.

### 6.5.2 Lower Bound for Optimization Time

In the previous section we found an upper bound for our Node-Based (1+1) EA. In this section we work on the lower bound analysis. In Section 6.5.2.1 we introduce an instance of the Euclidean GTSP, $G_S$, that is difficult to solve by means of our algorithm and discus some geometric properties of it. In Section 6.5.2.2 we show how the algorithm reaches a local optimum in our instance and discus how it can reach the global optimum after reaching the local optimum. Consequently we find a lower bound for the optimization time of the algorithm.

#### 6.5.2.1 A Hard Instance and its Geometric Properties

The hard instance presented in this section, which is partly illustrated in Figure 6.3, is composed of $m$ clusters. Let $a > 1$ be a constant. Only $\frac{m}{a}$ of these clusters have

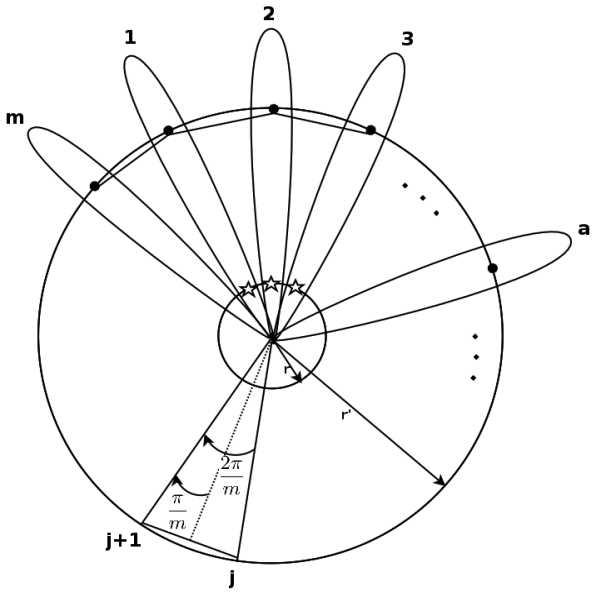


FIGURE 6.3: Euclidean hard instance, $G_S$, for Node-Based (1+1) EA

one node. Other clusters contain $m$ nodes which makes the total number of nodes $n = m(m - \frac{m}{a}) + \frac{m}{a}$. All nodes are connected to each other and the cost of travelling between them is their Euclidean distance.

In the clusters that have $m$ nodes, $m - 1$ nodes are placed on the small circle and are shown by a star in the picture. We refer to them as white nodes or inner nodes. For simplicity we assume that the inner nodes of each cluster all lie on the same position. The same result can be obtained by placing the nodes within a small circle having an arbitrarily small radius $\epsilon$. The remaining node of each cluster, shown black in the picture, is placed on the larger circle. Other $\frac{m}{a}$ clusters do not have any nodes on the small circle and have only one black node on the larger circle. The figure demonstrates how the clusters are distributed on the two circles. The arc between black nodes of two consecutive clusters subtend an angle of $\frac{2\pi}{m}$, while the arc between two consecutive one-node clusters subtend an angle of $a \cdot \frac{2\pi}{m}$.

If we represent the radius of inner and outer circles by $r$ and $r'$ respectively, then a black node and a white node have distance at least $r' - r$ and the length of edges between two adjacent black nodes is $2r' \sin(\frac{\pi}{m})$. The minimum length of edges between two black nodes of one-node clusters is also quite similar to previous formula with a greater angle: $2r' \sin(\frac{\pi}{\frac{m}{a}})$. We now prove that if

$$r < \frac{1}{2}\left(2\sin\left(\frac{\pi}{m}\right) - \sin\left(\frac{2\pi}{m}\right)\right) r' \tag{6.3}$$

then for $m \geq 8a$, the best tour on any spanning set that has at least one white node,

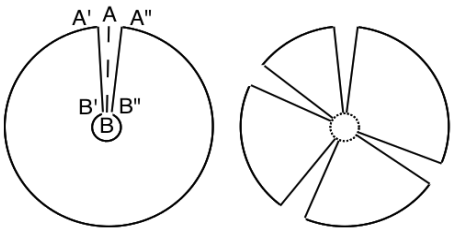


FIGURE 6.4: Left side: Case 1, Right side: Case 2

contains only 2 edges between outer and inner circles. We also show that the optimal solution consists of all the black nodes, but with high probability, the Node-Based (1+1) EA reaches a plateau of local optimums with $\frac{m}{a}$ black nodes and $m - \frac{m}{a}$ white nodes. Note that in such local optimums, selecting $\frac{m}{a}$ black nodes is a must, since there's no other choice for those clusters.

*Property* 6.16. The best tour on a spanning set that has at least one white node, contains only two edges between nodes on the inner and outer circle for $m \geq 8a$.

*Proof.* We first take into account the tour on a node set consisting of only the black nodes of one-node clusters. There is no other choice except selecting those nodes because their clusters have no other node. For such a node set, due to Theorem 2 of [105], the optimal tour is to visit all the nodes is the order they appear on the convex hull. This order will be respected in an optimal tour even if there are some inner nodes to visit as well, because according to Theorem 1 in [105] the optimal solution cannot intersect itself. In other words, if some white nodes are selected in the upper layer, while visiting the outer nodes with respect to their convex hull order, a solution occasionally travels the distance between outer circle and inner circle to visit some inner nodes, and then travels roughly the same distance back to the outer circle, to continue visiting the remaining outer nodes. As illustrated in Figure 6.4, this can be done generally in two ways:

1. Case 1: Leaving the outer circle only once and visiting all inner nodes together.
2. Case 2: Leaving the outer circle more than once and visiting some of the inner nodes each time.

We now show that there exists a solution for Case 1 that is less costly than all the solutions of Case 2. As a result, the best tour on a spanning set with at least one white node travels the distance between two circles only twice.

If we represent the number of times a tour leaves the outer circle to visit some nodes in the inner circle with $k$, then for the solutions of Case 1, $k = 1$ and for solutions of

Case 2, $k \geq 2$. For both cases the number of edges connecting the two circles is $2k$. The picture at the left side of Figure 6.4 illustrates a solution with $k = 1$ for which we find an upper bound of the tour cost as the following:

$$C(1) < 2\pi r' + 2\pi r + 2(r' - r) \tag{6.4}$$

The last part of this formula is two times of the length of edge $AB$ which is a direct line from the inner circle to the outer circle along their radius. The lengths of edges $A'B'$ and $A"B"$ are actually more than that because their ends are not from same clusters. Nevertheless, formula 6.4 presents an upper bound of the total cost of the tour, because we are considering the complete circumference of both circles. In other words, the distance between $A$ and $A'$ is included in the circumference of the large circle and the distance between $B$ and $B'$ is included in that of the small circle and according to quadrilateral inequality $|A'B'| < |A'A| + |AB| + |BB'|$.

On the other hand, a lower bound of the tour cost in all solutions of Case 2 is:

$$C(k) > \left(\frac{m}{a} - k\right) 2\sin\left(\frac{\pi}{m/a}\right) r' + 2k(r' - r) \tag{6.5}$$

In the above formula, $2\sin(\frac{\pi}{m/a})r'$ is the length of the edges connecting two consecutive clusters with one black node. These edges are the longest edges that can be removed from the tour when we add two edges connecting inner and outer circles. There are initially at least $m/a$ of these edges and in this formula we have omitted $k$ of them from the tour.

We can rewrite the right side of inequality 6.5 as:

$$C(k) > \left(\frac{m}{a}\right) 2\sin\left(\frac{\pi}{m/a}\right) r' - k \cdot 2\sin\left(\frac{\pi}{m/a}\right) r' + 2k(r' - r).$$

Since for $m > 8a$, $\sin(\frac{\pi}{m/a}) < 0.39$ and $(\frac{m}{a})\sin(\frac{\pi}{m/a}) > 3.06$ the above expression is at least:

$$2(3.06)r' - 2k \cdot 0.39r' + 2k(r' - r)$$

This expression is monotone increasing in $k$ when $r \leq 0.61r'$; therefore, setting $k = 2$ we get the smallest lower bound of $C(k)$ for $k \geq 2$:

$$C(k) > 4.56r' + 4(r' - r)$$

Now if we prove that the upper bound we found for $C(1)$ in Inequality 6.4 is less than the above expression, we can then conclude that $C(1) < C(k)$ for $k \geq 2$. Therefore, we

should prove that:

$$2\pi r' + 2\pi r + 2(r' - r) \leq 4.56r' + 4(r' - r)$$

$$\Leftrightarrow 2\pi r' + 2\pi r \leq 4.56r' + 2r' - 2r$$

$$\Leftrightarrow (\pi + 1)r \leq (-\pi + 3.28)r'$$

$$\Leftrightarrow r \leq \frac{-\pi + 3.28}{\pi + 1} r' \approx 0.033r'$$

The latest inequality holds, because the constraint we introduced on the value of $r$ in Equation 6.3 is quite tight and we can see that for $m \geq 8$ it gives us $r < 0.03r'$ which is a tighter bound for $r$ than what the right side of equation above gives us. □

*Property* 6.17. An optimal solution chooses all black nodes and visits them in clockwise or anti-clockwise order when $m \geq 7a$.

*Proof.* The tour comprising all black nodes has a cost strictly less than $2\pi r'$ which is the length of the circumference of the circle with radius $r'$. Therefore, we can state that $2\pi r'$ is an upper bound on the cost of the optimal solution. Besides, in Property 6.16 we saw that the best tour when at least one white node is selected has only two edges connecting the two circles. Therefore, as a lower bound on the cost of a solution with any spanning set other than all black nodes, we can use Formula 6.5 with $k = 1$ and get

$$C(1) \geq \left(\frac{m}{a} - 1\right) 2 \sin\left(\frac{\pi}{\frac{m}{a}}\right) r' + 2(r' - r).$$

We here show that with the assumptions we have on the value of $r$, this lower bound is greater than the upper bound we found for the cost of optimal solution. By replacing $r$ with its maximum value from Equation 6.3 we have:

$$C(1) \geq \left(2\left(\frac{m}{a} - 1\right) \sin\left(\frac{\pi}{\frac{m}{a}}\right) + 2 - \left(2 \sin\left(\frac{\pi}{m}\right) - \sin\left(\frac{2\pi}{m}\right)\right)\right) r'$$

since for $m \geq 7a$

$$\left(\frac{m}{a} - 1\right) \sin\left(\frac{\pi}{\frac{m}{a}}\right) > 2.60$$

and for $m \geq 4$

$$\left(2 \sin\left(\frac{\pi}{m}\right) - \sin\left(\frac{2\pi}{m}\right)\right) \leq 0.42$$

we can conclude that for $m \geq \max\{4, 7a\}$

$$C(1) \geq (2(2.60) + 2 - (0.42))r' = 6.78r' > 2\pi r'$$

As a result, for $m \geq 7a$ the minimum cost of such tours, is greater than $2\pi r'$ which is the maximum cost when all black nodes are selected. Hence the tour consisting of all black nodes is the optimal solution and since they comprise the convex hull the optimal Hamiltonian cycle on them would be visiting them in the order they appear in the convex hull. □

*Property* 6.18. Let $P$ and $P'$ be non-optimal spanning sets and $P_{out} \subset P$ and $P'_{out} \subset P'$ be their subset of outer nodes. Moreover, let $S$ and $S'$ be optimal solutions with respect to $P$ and $P'$ respectively. If $P_{out} \subset P'_{out}$ and $|P'_{out}| = |P_{out}| + 1$ then $C(S) < C(S')$.

*Proof.* The main idea behind this lemma is that distances in the inner circle are significantly shorter than distances in the large circle and if $r$ is sufficiently smaller than $r'$, any single mutation that replaces an inner node with the outer node of the same cluster, increases the cost of the whole tour.

According to Property 6.16, the permutation chosen in the lower layer, has all the inner nodes listed between two black nodes. If one inner node is removed and one outer node is added, the part of total tour that includes all inner nodes gets shorter and the part that connects black nodes gets longer. The maximum decrease for removing an inner node would be $4r$. In the following, we find the minimum increase for adding a black node.

We analyse the increase in two cases. The first case is illustrated in the left picture of Figure 6.5 in which the new black node is placed between two black nodes in the tour. $N$ is the new node and $M$ and $O$ are its neighbours. The edge connecting $M$ and $O$ will be removed from the tour and the two other edges in the triangle will be added. If we show the length of these edges by $C$, $A$ and $B$ respectively and the cost of the tour before and after this change by $C_{\text{old}}$ and $C_{\text{new}}$, then:

$$C_{\text{new}} = C_{\text{old}} - C + A + B$$

So the increase caused by this change would be:

$$d = C_{\text{new}} - C_{\text{old}} = A + B - C$$

By splitting $C$ with an orthogonal line from $N$ we can write $d$ as:

$$d = (A - C1) + (B - C2) \tag{6.6}$$

We claim that when $A$ and $B$ have their smallest values, $d$ has also its smallest value. We assume $A' \geq A$ and $B' \geq B$ and show that the corresponding $d'$ will be at least $d$.

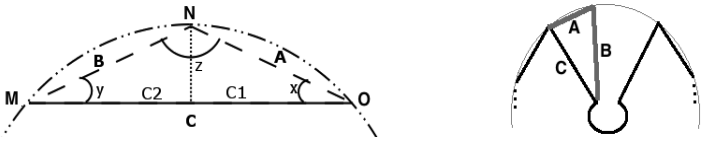


FIGURE 6.5: Left: Case 1, adding a new outer node between two outer nodes. Right: Case 2, adding a new outer node just before inner nodes

When $A' \geq A$ the arc between $O'$ and $N'$ is also greater than or equal to the arc between $O$ and $N$. Therefore, the angle $y'$, facing that arc, would be also greater than or equal to $y$. Besides, $C2 = B \cdot \cos(y)$, and all the things we said about $C2$ and $y$ hold for $C1$ and $x$ too. Altogether, we can write $d'$ as:

$$d' = A' - A' \cdot \cos(x') + B' - B' \cdot \cos(y')$$

since $y' \geq y$ and $x' \geq x$, and all of them are acute angles

$$d' \geq A' - A' \cdot \cos(x) + B' - B' \cdot \cos(y)$$

If we represent $A'$ by $\alpha A$ and $B'$ by $\beta B$ where $\alpha$ and $\beta$ are real numbers greater than one, then we have:

$$d' \geq \alpha \left(A - A \cdot \cos(x)\right) + \beta(B - B \cdot \cos(y))$$

$$\Rightarrow d' \geq \alpha(A - C1) + \beta(B - C2)$$

By comparing $d'$ with the value of $d$ in Equation 6.6 it holds that $d' \geq d$.

The shortest edges on the outer circle are from two consecutive clusters and have a length of $A = B = 2\sin(\pi/m)r'$. $C$ has similarly the value of $2\sin(2\pi/m)r'$. As a result, the minimum increase in the convex tour will be:

$$d = A + B - C = 4\sin\left(\frac{\pi}{m}\right)r' - 2\sin\left(\frac{2\pi}{m}\right)r' \tag{6.7}$$

The second case is when the new node is added just before or after visiting the inner nodes, as illustrated in the right picture of Figure 6.5. In this case, comparing to the previous case, edge $B$ is longer and the angle between $A$ and $B$ is closer to the right angle. The minimum length of $A$ is also the same as the minimum length of that in previous case. Altogether, with quite similar explanation to what we had for Case 1, the minimum increase in this case is larger than that in Case 1. Therefore the minimum increase in the convex tour, $d$, which is found in 6.7 is also less than the minimum increase in Case 2 and can be used for both cases.

On the other hand, as mentioned earlier the maximum decease caused by removing an inner node is $4r$. Therefore the total increase of the tour cost is at least

$$4\sin\left(\frac{\pi}{m}\right)r' - 2\sin\left(\frac{2\pi}{m}\right)r' - 4r$$

From our assumption on the value of $r$ in Equation 6.3 we can find that the above expression has a positive value; therefore, $C(S) < C(S')$. □

#### 6.5.2.2 Runtime Analysis

In this section, we give a lower bound on the runtime of Node-Based (1+1) EA. We start by presenting a Lemma about the initial solution that is chosen uniformly at random. Then using the Multiplicative Drift Theorem, which is presented in Section 4.5.2, we prove an upper bound for the time of reaching a locally optimal solution in Lemma 6.20. Then we discuss the main theorem of this section.

**Lemma 6.19.** *The initial solution with a spanning set that is chosen uniformly at random, has at least* $0.9\left(1-\frac{1}{a}\right)(m-1)$ *white nodes with probability* $1-e^{-\Omega(m)}$.

*Proof.* For $m-\frac{m}{a}$ clusters that have $m$ nodes, the probability of selecting one of white nodes is $\frac{m-1}{m}$. Therefore the expected number of selected white nodes is

$$E[X] = \left(m-\frac{m}{a}\right)\left(\frac{m-1}{m}\right) = \left(1-\frac{1}{a}\right)(m-1)$$

By Chernoff bounds we can have:

$$\text{Prob}\left(X < (0.9)(1-\frac{1}{a})(m-1)\right) \leq e^{-0.005(1-\frac{1}{a})(m-1)} = e^{-\Omega(m)}$$

Therefore, the probability that the initial solution has at least $(0.9)(1-\frac{1}{a})(m-1)$ white nodes is $1-e^{-\Omega(m)}$. □

**Lemma 6.20.** *Starting with an initial solution chosen uniformly at random, with probability* $1-e^{-\Omega(m)}$, *the Node-Based (1+1) EA reaches a local optimum on* $G_S$ *in expected time of* $O(m\ln m)$.

*Proof.* For a solution $x^{(t)}$ at time $t$, we define $X^{(t)}$ to be the number of $m$-node clusters for which the outer node is selected. Note that this function, as required in Theorem 4.6, maps the local optimum to zero and all other solutions to positive numbers.

If we assume that the number of $m$-node clusters that their outer node is chosen in solution $x^{(t)}$ is $k$, we can find the expected number of that for $x^{(t+1)}$ as follows:

As mentioned in Property 6.18, if only one mutation operation happens to increase the number of outer nodes, it will increase the cost and the algorithm will refuse it. Therefore, if only one mutation happens that is accepted by the algorithm, it has to change a node from the outer circle to the inner circle and decrease $X^t$ by 1. The probability of this event is at least

$$p_1 = k\left(\frac{1}{m}\right)\left(\frac{m-1}{m}\right)\left(1-\frac{1}{m}\right)^{m-1} \geq \frac{k}{m}\left(\frac{m-1}{m}\right)\left(\frac{1}{e}\right).$$

In the above formula, $\frac{1}{m}$ is the probability of mutation for any of the nodes in the spanning set and $\frac{m-1}{m}$ is the probability that the new selected node for the mutated cluster is a white node. We need one of $k$ clusters to mutate and all others to stay unchanged. In other words, $(1-\frac{1}{m})^{m-1}$ in the above formula is the probability of $m-1$ clusters to stay unchanged.

On the other hand, in some situations, some (one or more) mutations in the opposite direction can happen beside a mutation from outer circle to inner circle. If we want $X^t$ to increase by one, then at least two mutations must happen to change a node from inner circle to outer circle. The probability of this event is at most

$$p_{-1} = \frac{k}{m}\left(\frac{m-1}{m}\right)\binom{m-\frac{m}{a}-k}{2}\left(\frac{1}{m}\right)^2\left(\frac{1}{m}\right)^2 \leq \frac{k}{m}\left(\frac{1}{2!m^2}\right).$$

In the above formula, $(\frac{k}{m})(\frac{m-1}{m})$ is the probability of one node to change from outer circle to inner circle. Then we have the number of different ways we can select two clusters that their selected nodes lies on the inner circle. The first $(\frac{1}{m})^2$ is the probability that the 2 selected clusters mutate and the second $(\frac{1}{m})^2$ is the probability that after mutation the node on the outer circle is selected in those 2 clusters.

Generally, for $X^t$ to increase by $q$, we need at least one mutation from outer circle to inner circle and $q+1$ mutations in opposite direction and the probability of this even would be at most

$$\begin{aligned} p_{-q} &= \frac{k}{m}\left(\frac{m-1}{m}\right)\binom{m-\frac{m}{a}-k}{q+1}\left(\frac{1}{m}\right)^{q+1}\left(\frac{1}{m}\right)^{q+1} \\ &\leq \frac{k}{m}\left(\frac{1}{(q+1)!m^{q+1}}\right) \end{aligned}$$

As a result, the difference made in $X^t$ by the next step would be at least

$$E[X^{(t)} - X^{(t+1)} \mid X^{(t)} = k] \geq p_1 - \sum_{q=1}^{m} q \cdot p_{-q}$$

By replacing the lower bound of $p_1$ and upper bounds of $p_{-q}$, we get

$$\begin{aligned} E[[X^{(t)} - X^{(t+1)} \mid X^{(t)} = k] &\geq \frac{k}{m}\left(\frac{m-1}{m}\right)\left(\frac{1}{e}\right) - \frac{k}{m}\left(\frac{1}{2!m^2}\right) - \dots \\ &\quad -m\frac{k}{m}\left(\frac{1}{(m+1)!m^{m+1}}\right) \\ &\geq \frac{k}{m}\left(\frac{m-1}{em} - \frac{1}{m^2} - \dots - \frac{1}{m^{m+1}}\right) \\ &\geq \frac{k}{m}\left(\frac{m-1}{em} - m \cdot \frac{1}{m^2}\right) \\ &\geq \frac{k}{m}\left(\frac{m-1-e}{em}\right) \end{aligned}$$

For $m \geq 4$ the expression $(\frac{m-1-e}{em})$ is at least $\frac{3-e}{4e}$. So setting $\delta = \frac{3-e}{4em}$ and using the Multiplicative Drift Theorem (Theorem 4.6) we find the expected time of reaching the local optimum as:

$$E[T \mid X^{(0)} = 0.1m] \leq \frac{1 + \ln(0.1m/1)}{\frac{3-e}{3em}} = O(m \ln m)$$

In the above formula we have assumed $X^{(0)} \leq 0.1m$ because from Lemma 6.19 we know that with probability $1 - e^{-\Omega(m)}$, the initial solution has less than $0.1m$ black nodes other than the fixed black nodes. □

**Theorem 6.21.** *Starting with an initial solution chosen uniformly at random, if $m \geq 8a$, then the optimization time of the Node-Based (1+1) EA presented in Algorithm 14 on $G_S$ is $\Omega\left((\frac{n}{2})^{m-\frac{m}{a}}\right)$ with probability $1 - e^{-\Omega(m^\delta)}$, $\delta > 0$.*

*Proof.* In order to prove this theorem, we introduce a phase $P$ in which

1. The algorithm reaches a local optimum with high probability
2. The algorithm does not reach the global optimum with high probability

Then we show that after this phase, only a direct jump from the local optimum to the global optimum will help the algorithm improve the results, probability of which is $\left(\frac{1}{m^2}\right)^{m-\frac{m}{a}}$.

As we saw in Lemma 6.20, the expected time of Node-Based (1+1) EA to reach the local optimum is $O(m \ln m)$. Let $c$ be the appropriate constant, so that $c \cdot m \ln m$ is the expected time for reaching that local optimum. Now consider a phase of $2c \cdot m \ln m$

steps. If $T$ is the actual time at which the local optimum is reached, by Markov's inequality (Section 4.2.1) we have: $\text{Prob}(T > 2c \cdot m \ln m) \leq \frac{1}{2}$. If we repeat this phase for $\frac{m^\varepsilon}{\ln m}$ times, $\varepsilon > 0$ a constant, then we get a phase of $P = 2c \cdot m^{1+\varepsilon}$ steps in which the probability of not reaching the local optimum is:

$$\text{Prob}(T > 2c \cdot m^{1+\varepsilon}) \leq \left(\frac{1}{2}\right)^{-\frac{m^\varepsilon}{\ln m}} = e^{-\Omega(m^\delta)},$$

where $0 < \delta < \varepsilon$. As a result, the algorithm reaches the local optimum in phase $P$ with probability $1 - e^{-\Omega(m^\delta)}$. We here prove that in this phase, the algorithm does not reach the global optimum with probability $1 - e^{-\Omega(m^\varepsilon)}$.

From Lemma 6.19 we know that with high probability the initial solution has not too many black nodes other than the fixed black nodes. Here we show that with high probability the number of these nodes does not increase significantly during the phase $P$; hence, the global optimum will not be reached. The probability of selecting each of the clusters for a mutation is $1/m$ and for clusters with $m$ nodes, the probability of changing the selected node to the black node is $\frac{1}{m}$; therefore, at each step, the probability that each cluster's node is changed from one of its inner nodes to its outer node is $\frac{1}{m^2}$. For $m - \frac{m}{a}$ clusters, at each step the expected number of clusters that face such a mutation is at most $\frac{1}{m}$ and in a phase of $2cm^{1+\varepsilon}$ steps, is $2cm^\varepsilon$. If we define $X$ as the number of clusters that will have a mutation like this, then by Chernoff bound (Section 4.2.2) we have

$$\text{Prob}(X \geq 3cm^\varepsilon) \leq e^{-2cm^\varepsilon(0.5)^2/3} = e^{-\Omega(m^\varepsilon)}.$$

Therefore, with high probability, during the mentioned phase, at most $3cm^\varepsilon$ clusters will happen to have a mutation with the result of selecting their black node. Besides, from Lemma 6.19 we know that with probability $1 - e^{-\Omega(m)}$, the initial solution has at least $0.9(1 - \frac{1}{a})(m-1)$ white nodes. Hence, with probability $e^{-\Omega(m^\varepsilon)}$, the algorithm will not reach a state with less than $0.9(1 - \frac{1}{a})(m-1) - 3cm^\varepsilon$ white nodes during phase $P$. As a result, the probability of having a direct jump to the global optimum in phase $P$ is at most

$$2c \cdot m^{1+\varepsilon} \left(\frac{1}{m^2}\right)^{0.9(1-\frac{1}{a})(m-1)-3cm^\varepsilon} = m^{-\Omega(m)}.$$

Consequently, with high probability, the global optimum will not be reached during phase $P$. According to Property 6.18, no mutation from the inner circle to the outer circle can decrease the tour cost when the resulting solution is not the optimal solution. Hence, such a change may only be accepted by the algorithm when another mutation on the other direction happens at the same step. At the local optimum, there is no black node other than the fixed black nodes and no mutation from the outer circle to

the inner circle can happen; therefore, a mutation from the inner circle to the outer circle can not happen either. As a result, after reaching a local optimum, only a direct jump to the global optimum can help moving towards the global optimum and the probability of such a jump is $\left(\frac{1}{m^2}\right)^{m-\frac{m}{a}}$. We now consider $(\frac{m^2}{2})^{m-\frac{m}{a}}$ steps following phase $P$. The probability of reaching the optimum solution is by union bound at most: $\left(\frac{m^2}{2}\right)^{m-\frac{m}{a}}\left(\frac{1}{m^2}\right)^{m-\frac{m}{a}} = (\frac{1}{2})^{m-\frac{m}{a}}$. Hence the probability of not reaching the global optimum in the mentioned phase is $1-(\frac{1}{2})^{m-\frac{m}{a}} = 1-e^{-\Omega(m)}$. Altogether, with probability $1-e^{-\Omega(m^\delta)}$, the optimization time is at least $\left(\frac{m^2}{2}\right)^{m-\frac{m}{a}}$. □

## 6.6 Conclusion

Evolutionary algorithms and local search approaches have been shown to be very successful for solving the generalized travelling salesman problem. In this chapter we have investigated two common hierarchical representations together with simple evolutionary algorithms from a theoretical perspective. We prove lower and upper bounds for optimization time of (1+1) EA with both approaches. Our analyses show that the Cluster-Based (1+1) EA is a fixed parameter evolutionary algorithm with respect to the number of clusters, while the same thing does not hold for the Node-Based (1+1) EA. However, we have proved that there are instances which Node-Based (1+1) EA solves to optimality in polynomial time, while Cluster-Based (1+1) EA needs exponential time to find an optimal solution for them.

In order to prove an exponential lower bound for the optimization time of the Node-Based (1+1) EA, we have presented a Euclidean instance of the GTSP. Our lower bound analysis for this geometric instance shows that the Euclidean case is hard to solve even if we assume that the lower layer TSP is solved to optimality in no time.

# Chapter 7

# Multi-Objective Evolutionary Algorithms for the Weighted Vertex Cover Problem

## 7.1 Introduction

In this chapter, we consider the weighted vertex cover problem (Definition 3.4 in Section 3.3) with integer weights on the nodes, where the goal is to find a vertex cover of minimum weight. We extend the investigations carried out in [77] to this problem. In [77], multi-objective models in combination with a simple multi-objective evolutionary algorithm called Global SEMO are investigated. They have shown that Global SEMO, with a problem specific mutation operator is a fixed parameter evolutionary algorithm for the classical vertex cover problem and finds 2-approximations in expected polynomial time. Kratsch and Neumann [77] have also introduced an alternative mutation operator and have proved that Global SEMO using this mutation operator finds a $(1+\varepsilon)-$approximation in expected time $O(n^2 \log n + OPT \cdot n^2 + n \cdot 4^{(1-\varepsilon)OPT})$, where $OPT$ is the cost of the optimal solution. One key argument for the results presented for the classical vertex cover problem is that the population size is always upper bounded by $n+1$. This argument does not hold in the weighted case. Therefore, we study how a variant of Global SEMO using appropriate diversity mechanisms is able to deal with the weighted vertex cover problem.

The focus of this chapter is on finding good approximations of an optimal solution. The time complexity analysis is performed with respect to $n$, $W_{max}$, and $OPT$, which denote the number of vertices, the maximum weight in the input graph, and the cost of

the optimal solution respectively. We first study the expected time until Global SEMO with standard mutation operator has found a 2-approximation in dependence of $n$ and $OPT$. Afterwards, we consider DEMO, a variant of Global SEMO, which incorporates $\varepsilon$-dominance (Section 2.4.2) as a diversity mechanism. It is shown that DEMO finds a 2-approximation in expected polynomial time.

The work of this chapter is based on parts of two papers, which were written in collaboration with other authors. One of the papers has been presented at conference PPSN'16, and the other is submitted to a journal [103, 104]. The outline of this chapter is as follows. In Section 7.2, the problem definition is presented as well as the classical Global SEMO algorithm and DEMO algorithm. Runtime analysis for Global SEMO for finding a 2-approximation is presented in Section 7.3. Section 7.4 includes the analysis that shows DEMO can find 2-approximations of the optimum in expected polynomial time. At the end, in Section 7.5 we summarize and conclude.

## 7.2 Preliminaries

We consider the weighted vertex cover problem defined in Definition 3.4 of Section 3.3. We assume that all the weights are integers and consider the standard node-based approach, i.e. the search space is $\{0,1\}^n$ and for a solution $x = (x_1, \ldots, x_n)$ the node $v_i$ is chosen iff $x_i = 1$.

As discussed in Section 3.3.2, this problem can be formulated as an ILP problem and by relaxing the constraint $x_i \in \{0,1\}$ to $x_i \in [0,1]$, the LP formulation of the fractional weighted vertex cover is obtained. Hochbaum [62] has shown that we can find a 2-approximation using the LP result of the relaxed weighted vertex cover. This can be done by including any vertex $v_i$ for which $x_i \geq \frac{1}{2}$.

We consider primarily multi-objective approaches for the weighted vertex cover problem. Given a multi-objective fitness function $f = (f_1, \ldots, f_d)\colon S \to \mathbb{R}$ where all $d$ objectives should be minimized, we have $f(x) \leq f(y)$ iff $f_i(x) \leq f_i(y)$, $1 \leq i \leq d$. We say that $x$ (weakly) dominates $y$ iff $f(x) \leq f(y)$. Furthermore, we say that $x$ (strongly) dominates $y$ iff $f(x) \leq f(y)$ and $f(x) \neq f(y)$.

We now introduce the objectives used in our multi-objective evolutionary algorithm. Let $G(x)$ be the graph obtained from $G$ by removing all nodes chosen by $x$ and the corresponding covered edges. Formally, we have $G(x) = (V(x), E(x))$ where $V(x) = V \setminus \{v_i \mid x_i = 1\}$ and $E(x) = E \setminus \{e \mid e \cap (V \setminus V(x)) \neq \emptyset\}$. Kratsch and Neumann [77]

**Algorithm 16:** Global SEMO

```
Choose x ∈ {0,1}^n uniformly at random;
Determine f(x);
P ← {x};
repeat
    Choose x ∈ P uniformly at random;
    Create x' by flipping each bit x_i of x with probability 1/n;
    Determine f(x');
    if ∄y ∈ P | f(y) ≤ f(x') then
        P ← {x'};
        delete all other solutions z ∈ P with f(x') ≤ f(z) from P;
until termination condition satisfied;
```

investigated a multi-objective baseline algorithm called Global SEMO using the LP-value for $G(x)$ as one of the fitness values for the (unweighted) minimum vertex cover problem.

Our goal is to expand the analysis on behaviour of multi-objective evolutionary algorithms to the weighted vertex cover problem. In order to do this, we modify the fitness function that was used in Global SEMO in [77], to match the weighted version of the problem. We investigate the multi-objective fitness function $f(x) = (Cost(x), LP(x))$, where

- $Cost(x) = \sum_{i=1}^{n} w(v_i)x_i$ is the sum of weights of selected vertices
- $LP(x)$ is the value of an optimal solution of the LP for $G(x)$.

We analyse Global SEMO (Algorithm 16) with this fitness function using the standard mutation operator flipping each bit with probability $1/n$.

In the fitness function used in Global SEMO, both $Cost(x)$ and $LP(x)$ can be exponential with respect to the input size; therefore, we need to deal with exponentially large number of solutions, even if we only keep the Pareto front. One approach for dealing with this problem is using the concept of $\varepsilon-$dominance described in Section 2.4.2. Recall that in this approach, the objective space is partitioned into a polynomial number of boxes and at most one solution from each box is kept in the population.

Motivated by this approach, DEMO (Diversity Evolutionary Multi-objective Optimizer) has been investigated in [89, 90]. In Section 7.4, we analyze DEMO (Algorithm 17) in which only one non-dominated solution can be kept in the population for each box based on a predefined criteria. In our setting, among two solutions $x$ and $y$ from one box, $y$ is kept in $P$ and $x$ is discarded if $Cost(y) + 2 \cdot LP(y) \leq Cost(x) + 2 \cdot LP(x)$.

**Algorithm 17:** DEMO

```
1  Choose x ∈ {0,1}^n uniformly at random;
2  Determine b(x);
3  P ← {x};
4  repeat
5      Choose x ∈ P uniformly at random;
6      Create x' by flipping each bit x_i of x with probability 1/n;
7      Determine f(x') and b(x');
8      if ∃y ∈ P | (f(y) ≤ f(x') ∧ f(y) ≠ f(x')) ∨ (b(y) = b(x') ∧ Cost(y) + 2 · LP(y) ≤
       Cost(x') + 2 · LP(x')) then
9          Go to 4;
10     else
11         P ← {x'};
12         delete all other solutions z ∈ P where f(x') ≤ f(z) ∨ b(z) = b(x') from P;
13 until termination condition satisfied;
```

To implement the concept of $\varepsilon-$dominance in DEMO, we use the parameter $\delta = \frac{1}{2n}$ and define the boxing function $b : \{0,1\}^n \to \mathbb{N}^2$ as:

$$\begin{aligned} b_1(x) &= \lceil \log_{1+\delta}(1 + Cost(x)) \rceil, \\ b_2(x) &= \lceil \log_{1+\delta}(1 + LP(x)) \rceil, \end{aligned}$$

Analysing the runtime of our evolutionary algorithms, we are interested in the expected number of rounds of the repeat loop until a solution of desired quality has been obtained. We call this the expected time until the considered algorithm has achieved its desired goal.

## 7.3 Analysis of Global SEMO

In this section we analyse the expected time of Global SEMO to find approximations for the weighted vertex cover problem in dependence of the input size and OPT. Before we present our analysis for Global SEMO, we state some basic properties of the solutions in our multi-objective model. The following theorem shown by Balinski [7] states that all basic feasible solutions of the fractional vertex cover, which are the extremal points or the corner solutions of the polyhedron that forms the feasible space, are half-integral.

**Theorem 7.1.** *Each basic feasible solution $x$ of the relaxed vertex cover ILP is half-integral, i.e., $x \in \{0, 1/2, 1\}^n$. [7]*

As a result of Theorem 7.1, there always exists a half integral optimal LP solution for a vertex cover problem. In several parts of this paper, we make use of this result. We

establish the following two lemmata which we will use later on in the analysis of our algorithms.

**Lemma 7.2.** *For any $x \in \{0,1\}^n$, $LP(x) \leq LP(0^n) \leq OPT$.*

*Proof.* Let $y$ be the LP solution of $LP(0^n)$. Also, for any solution $x$, let $G(x)$ be the graph obtained from $G$ by removing all vertices chosen by $x$ and their edges. The solution $0^n$ contains no vertices; therefore, $y$ is the optimal fractional vertex cover for all edges of the input graph. Thus, for any solution $x$, $y$ is a (possibly non-optimal) fractional cover for $G(x)$; therefore, $LP(x) \leq LP(0^n)$. Moreover, we have $LP(0^n) \leq OPT$ as $LP(0^n)$ is the optimal value of the LP relaxation. □

**Lemma 7.3.** *Let $x = \{x_1, \cdots, x_n\}, x_i \in \{0,1\}$ be a solution and $y = \{y_1, \cdots, y_n\}, y_i \in [0,1]$ be a fractional solution for $G(x)$. If there is a vertex $v_i$ where $y_i \geq \frac{1}{2}$, mutating $x_i$ from 0 to 1 results in a solution $x'$ for which $LP(x') \leq LP(x) - y_i \cdot w(v_i) \leq LP(x) - \frac{1}{2}w(v_i)$.*

*Proof.* The graph $G(x')$ is the same as $G(x)$ excluding the edges connected to $v_i$. Therefore, the solution $y' = \{y_1, \cdots, y_{i-1}, 0, y_{i+1}, y_n\}$ is a fractional vertex cover for $G(x')$ and has a cost of $LP(x) - y_i w(v_i)$. The cost of the optimal fractional vertex cover of $G(x')$ is at most as great as the cost of $y'$; thus $LP(x') \leq LP(x) - y_i w(v_i) \leq LP(x) - \frac{1}{2}w(v_i)$. □

We now analyse the runtime behaviour of Global SEMO (Algorithm 16) with the standard mutation operator, in dependence of OPT. For our analysis, we start by giving an upper bound on the population size of Global SEMO. Then we consider the expected time of Global SEMO to reach a population which contains the empty set of nodes. Once included, such a solution will never be removed from the population as it is minimal with respect to the cost function. At the end of this section, in Theorem 7.6 we present a pseudo polynomial upper bound on the expected optimization time of Global SEMO.

**Lemma 7.4.** *The population size of Algorithm 16 is upper bounded by $2 \cdot OPT + 1$.*

*Proof.* For any solution $x$ there exists an optimal fractional vertex cover which is half-integral (Theorem 7.1). Moreover, we are assuming that all the weights are integer values. Therefore, $LP(x)$ can only take $2LP(0^n) + 1$ different values, because $LP(0^n)$ is an upper bound on $LP(x)$ (Lemma 7.2). For each value of $LP$, only one solution is in $P$, because Algorithm 16 keeps non-dominated solutions only. Therefore, the population size of this algorithm is upper bounded by $2 \cdot LP(0^n) + 1$ which is at most $2 \cdot OPT + 1$ due to Lemma 7.2. □

**Lemma 7.5.** *The search point $0^n$ is included in the population in expected time of $O(OPT \cdot n(\log W_{max} + \log n))$.*

*Proof.* From Lemma 7.4 we know that the population contains at most $2 \cdot OPT + 1$ solutions. Therefore, at each step, there is a probability of $\frac{1}{2 \cdot OPT+1}$ that the solution $x_{min}$ is selected where $Cost(x_{min}) = \min_{x \in P} Cost(x)$.

If $Cost(x_{min}) > 0$, there must be $k \geq 1$ vertex such as $v_i$ in $x_{min}$ where $x_i = 1$. Let $\Delta^t$ be the improvement that happens on the minimum cost in $P$ at step $t$. If all the 1-bits in solution $x_{min}$ flip to zero, at the same step or different steps, a solution $0^n$ will be obtained with $Cost(0^n) = 0$, which implies that the expected improvement that flipping each 1-bit makes is $\Delta^t = \frac{Cost(x_{min})}{k}$ at each step $t$. Note that flipping 1-bits always improves the minimum cost and the new solution is added to the population. Moreover, flipping the 0-bits does not improve the minimum cost in the population and $x_{min}$ is not replaced with the new solution in that case.

At each step, with probability $\frac{1}{e}$ only one bit flips. With probability $\frac{k}{n}$, the flipping bit is a 1-bit, and makes an expected improvement of $\Delta^t = \frac{Cost(x_{min})}{k}$, and with probability $1 - \frac{k}{n}$, a 0-bit is flipped with $\Delta^t = 0$. We can conclude that the expected improvement of minimum cost, when only one bit of $x_{min}$ flips, is

$$\frac{k}{n} \cdot \frac{Cost(x_{min})}{k} = \frac{Cost(x_{min})}{n}$$

Moreover, the algorithm selects $x_{min}$ and flips only one bit with probability $\frac{1}{(2 \cdot OPT+1) \cdot e}$; therefore, the expected improvement of minimum cost is

$$E[\Delta^t \mid x_{min}] \geq \frac{Cost(x_{min})}{(2 \cdot OPT + 1) \cdot e \cdot n}$$

The maximum value that $Cost(x_{min})$ can take is bounded by $W_{max} \cdot n$, and for any solution $x \neq 0^n$, the minimum value of $Cost(x)$ is at least 1. Using Multiplicative Drift Analysis (Section 4.5.2) with $s_0 \leq W_{max} \cdot n$ and $s_{min} \geq 1$, we can conclude that in expected time $O\left(OPT \cdot n(\log W_{max} + \log n)\right)$ solution $0^n$ is included in the population. □

We now show that Global SEMO is able to achieve a 2-approximation efficiently as long as OPT is small.

**Theorem 7.6.** *The expected number of iterations of Global SEMO until the population $P$ contains a 2-approximation is $O(OPT \cdot n(\log W_{max} + \log n))$.*

*Proof.* Let $x$ be a solution that minimizes $LP(x)$ under the constraint that $Cost(x) + 2 \cdot LP(x) \leq 2 \cdot OPT$. Note that this constraint holds for solution $0^n$ since $LP(0^n) \leq OPT$, and according to Lemma 7.5, solution $0^n$ exists in the population in expected time of $O\left(OPT \cdot n(\log W_{max} + \log n)\right)$.

If $LP(x) = 0$, then all edges are covered and $x$ is a 2-approximate vertex cover, because we have $Cost(x) + 2 \cdot LP(x) \leq 2 \cdot OPT$ as the constraint. Otherwise, some edges are uncovered and any LP solution of $G(x)$ assigns at least $\frac{1}{2}$ to at least one vertex of any uncovered edge. Let $y = \{y_1, \cdots, y_n\}$ be a basic LP solution for $G(x)$. According to Theorem 7.1, $y$ is a half-integral solution.

Let $\Delta^t$ be the improvement that happens on the minimum $LP$ value among solutions that fulfil the constraint at time step $t$. Also, let $k$ be the number of nodes that are assigned at least $\frac{1}{2}$ by $y$. Flipping only one of these nodes by the algorithm happens with probability at least $\frac{k}{e \cdot n}$. According to Lemma 7.3, flipping one of these nodes, $v_i$, results in a solution $x'$ with $LP(x') \leq LP(x) - \frac{1}{2}w(v_i)$. Observe that the constraint of $Cost(x') + 2 \cdot LP(x') \leq 2 \cdot OPT$ holds for solution $x'$. Therefore, $\Delta^t \geq y_i \cdot w(v_i)$, which is in expectation at least $\frac{LP(x)}{k}$ due to definition of $LP(x)$. Moreover, at each step, the probability that $x$ is selected and only one of the $k$ bits defined above flips is $\frac{k}{(2 \cdot OPT+1) \cdot e \cdot n}$. As a result we have:

$$E[\Delta^t \mid x] \geq \frac{k}{(2 \cdot OPT + 1) \cdot e \cdot n} \cdot \frac{LP(x)}{k} = \frac{LP(x)}{en(2 \cdot OPT + 1)}$$

According to Lemma 7.2 for any solution $x$, we have $LP(x) \leq OPT$. We also know that for any solution $x$ which is not a complete cover, $LP(x) \geq 1$, because the weights are positive integers. Using the method of Multiplicative Drift Analysis [34] with $s_0 \leq OPT$ and $s_{min} \geq 1$, in expected time of $O(OPT \cdot n \log OPT)$ a solution $y$ with $LP(y) = 0$ and $Cost(y) + 2LP(y) \leq 2OPT$ is obtained which is a 2-approximate vertex cover. Overall, since we have $OPT \leq W_{max} \cdot n$, the expected time of finding this solution is $O(OPT \cdot n(\log W_{max} + \log n))$. □

## 7.4 Analysis of DEMO

Due to Lemma 7.4, with Global SEMO, the population size is upper bounded by $O(OPT)$, which can be exponential in terms of the input size. In this section, we analyse the other evolutionary algorithm, DEMO (Algorithm 17), that uses some diversity handling mechanisms for dealing with exponentially large population sizes. The following lemmata are used in the proof of Theorem 7.10.

**Lemma 7.7.** *Let $W_{max}$ be the maximum weight assigned to a vertex. The population size of DEMO is upper bounded by $O\left(n \cdot (\log n + \log W_{max})\right)$.*

*Proof.* The values that can be taken by $b_1$ are integer values between 0 and $\lceil \log_{1+\delta}(1 + Cost(1^n)) \rceil$ and the values that can be taken by $b_2$ are integer values between 0 and $\lceil \log_{1+\delta}(1 + LP(0^n)) \rceil$ (Lemma 7.2). Since $n \cdot W_{max}$ is an upper bound for both $Cost(1^n)$ and $LP(0^n)$, the number of rows and also the number of columns are bounded by

$$\begin{aligned} k &= \left(1 + \lceil \log_{1+\delta}(1 + n \cdot W_{max}) \rceil\right) \\ &\leq \left(1 + \left\lceil \frac{\log(1 + n \cdot W_{max})}{\log(1+\delta)} \right\rceil\right) \\ &= O\left(n \cdot (\log n + \log W_{max})\right) \end{aligned}$$

The last equality holds because $\delta = \frac{1}{2n}$.

We here show that the size of the population is $P_{size} \leq 2k - 1$. Since the dominated solutions according to $f$ are discarded by the algorithm, none of the solutions in $P$ can be located in a box that is dominated by another box that contains a solution in $P$. Moreover, at most one solution from each box is kept in the population; therefore, $P_{size}$ is at most the maximum number of boxes where none of them dominates another.

Let $k_1$ be the number of boxes that contain a solution of $P$ in the first column. Let $r_1$ be the smallest row number among these boxes. Observe that $r_1 \leq k - k_1 + 1$ and the equality holds when the boxes are from rows $k$ down to $k - k_1 + 1$. Any box in the second column with a row number of $r_1 + 1$ or above is dominated by the box of the previous column and row $r_1$. Therefore, the maximum row number for a box in the second column, that is not dominated, is $r_1 \leq k - k_1 + 1$. With generalizing the idea, the maximum row number for a box in the column $i$, that is not dominated, is $r_{i-1} \leq k - k_1 - \cdots - k_{i-1} + i - 1$, where for $1 \leq j \leq k$, $k_j$ is the number of boxes that contain a solution of $P$ in column $j$.

The last column has $k_k \leq r_{k-1}$ boxes which gives us:

$$k_k \leq r_{k-1} \leq k - k_1 - \cdots - k_{k-1} + k - 1$$

This implies that

$$k_1 + \cdots + k_k \leq r_{k-1} \leq 2k - 1$$

which completes the proof. □

**Lemma 7.8.** *The search point $x_z = 0^n$ is included in the population in expected time of $O(n^3(\log n + \log W_{max})^2)$.*

*Proof.* From Lemma 7.7 we know that the population contains $P_{size} = O\left(n \cdot (\log n + \log W_{max})\right)$ solutions. Therefore, at each step, there is a probability of at least $\frac{1}{p_{size}}$ that the solution $x_{min}$ is selected where $b_1(x_{min}) = \min_{x \in P}\ b_1(x)$.

If $b_1(x_{min}) = 0$, we have $Cost(x_{min}) = 0$, which means $x_{min} = 0^n$ since the weights are greater than 0.

If $b_1(x_{min}) \neq 0$, there must be at least one vertex $v_i$ in $x_{min}$ where $x_i = 1$. Consider $v_j$ the vertex that maximizes $w(v_i)$ among vertices $v_i$ where $x_i = 1$. If $Cost(x) = C$, then $w(v_j) \geq \frac{C}{n}$, because $n$ is an upper bound on the number of vertices selected by $x_{min}$. As a result, removing vertex $x_j$ from solution $x_{min}$ results in a solution $x'$ for which $Cost(x') \leq C \cdot (1 - \frac{1}{n})$. Using this value of $Cost(x')$, we have

$$\begin{aligned}
(1+\delta)(1+Cost(x')) &\leq 1 + \delta + C(1 - \frac{1}{n})(1+\delta) \\
&\leq 1 + \delta + C + C(\delta - \frac{1}{n} - \frac{\delta}{n}) \\
&\leq 1 + C\delta + C + C(\delta - \frac{1}{n} - \frac{\delta}{n}) \\
&\leq 1 + C + C(2\delta - \frac{1}{n} - \frac{\delta}{n}) \\
&\leq 1 + C
\end{aligned}$$

The third inequality above holds because $C \geq 1$ and the last one holds because $\delta = \frac{1}{2n}$. From $(1+\delta)(1+Cost(x')) \leq 1 + C$ we observe that

$$1 + \log_{1+\delta}(1 + Cost(x')) \leq \log_{1+\delta}(1 + C)$$

which implies $b_1(x') \leq b_1(x) - 1$. Note that $x'$ is obtained by performing a 1-bit flip on $x$ and is done at each step with a probability of at least

$$\frac{1}{P_{size}} \cdot \frac{1}{n} \cdot (1 - \frac{1}{n})^{n-1}$$

$$= \Omega\left(\frac{1}{n(\log n + \log W_{max})} \cdot \frac{1}{n}\right)$$

Therefore, in expected time of at most $O\left(n^2(\log n + \log W_{max})\right)$ the new solution, $x'$ is obtained which is accepted by the algorithm because it is placed in a box with a smaller value of $b_1$ than all solutions in $P$ and hence not dominated. There are $O\left(n(\log n + \log W_{max})\right)$ different values for $b_1$; therefore, the solution $x_z = 0^n$ with $b_1(x_z) = 0$ is found in expected time of at most $O\left(n^3(\log n + \log W_{max})^2\right)$. □

**Lemma 7.9.** *Let $x \in P$ be a search point such that $Cost(x) + 2 \cdot LP(x) \leq 2 \cdot OPT$ and $b_2(x) > 0$. There exists a 1-bit flip leading to a search point $x'$ with $Cost(x') + 2 \cdot LP(x') \leq 2 \cdot OPT$ and $b_2(x') < b_2(x)$.*

*Proof.* Let $y = \{y_1 \cdots y_n\}$ be a basic half integral LP solution for $G(x)$. Since $b_2(x) = LP(x) \neq 0$, there must be at least one uncovered edge; hence, at least one vertex $v_i$ has a $y_i \geq \frac{1}{2}$ in LP solution $y$. Consider $v_j$ the vertex that maximizes $y_i w(v_i)$ among vertices $v_i$, $1 \leq i \leq n$. Also, let $x'$ be a solution obtained by adding $v_j$ to $x$. Since solutions $x$ and $x'$ are only different in one vertex, $v_j$, we have $Cost(x') = Cost(x) + w(v_j)$. Moreover, according to Lemma 7.3, $LP(x') \leq LP(x) - \frac{1}{2} \cdot w(v_j)$. Therefore,

$$Cost(x') + 2 \cdot LP(x') \leq Cost(x) + w(v_j) + 2\left(LP(x) - \frac{w(v_j)}{2}\right)$$

$$\leq Cost(x) + 2 \cdot LP(x) \leq 2 \cdot OPT$$

which means solution $x'$ fulfils the mentioned constraint. If $LP(x) = W$, then $y_j w(v_j) \geq \frac{W}{n}$, because $n$ is an upper bound on the number of vertices selected by the LP solution. As a result, using Lemma 7.3, we get $LP(x') \leq W \cdot (1 - \frac{1}{n})$. Therefore, with similar analysis as Lemma 7.8 we get:

$$\begin{aligned}(1+\delta)\left(1 + LP(x')\right) &\leq 1 + \delta + W\left(1 - \frac{1}{n}\right)(1+\delta) \\ &\leq 1 + W\end{aligned}$$

This inequality implies

$$1 + \log_{1+\delta}(1 + LP(x')) \leq \log_{1+\delta}(1 + W)$$

As a result, $b_2(x') < b_2(x)$ holds for $x'$, which is obtained by performing a 1-bit flip on $x$, and the lemma is proved. □

**Theorem 7.10.** *The expected time until DEMO constructs a 2-approximation vertex cover is $O\left(n^3 \cdot (\log n + \log W_{max})^2\right)$.*

*Proof.* Consider solution $x \in P$ that minimizes $b_2(x)$ under the constraint that $Cost(x) + 2 \cdot LP(x) \leq 2 \cdot OPT$. Note that $0^n$ fulfils this constraint and according to Lemma 7.8, the solution $0^n$ will be included in $P$ in time $O\left(n^3(\log n + \log W_{max})^2\right)$.

If $b_2(x) = 0$ then $x$ covers all edges and by selection of $x$ we have $Cost(x) \leq 2 \cdot OPT$, which means that $x$ is a 2-approximation.

In case $b_2(x) \neq 0$, according to Lemma 7.9 there is a one-bit flip on $x$ that results in a new solution $x'$ for which $b_2(x') < b_2(x)$, while the mentioned constraint also holds for it. Since the population size is $O(n \cdot (\log n + \log W_{max}))$ (Lemma 7.7), this 1-bit flip happens with a probability of $\Omega\left(n^{-2} \cdot (\log n + \log W_{max})^{-1}\right)$ and $x'$ is obtained in expected time of $O(n^3 \cdot (\log n + \log W_{max})^2)$. This new solution will be added to $P$ because a solution $y$ with $Cost(y) + 2 \cdot LP(y) > 2 \cdot OPT$ can not dominate $x'$ with $Cost(x') + 2 \cdot LP(x') \leq 2 \cdot OPT$, and $x'$ has the minimum value of $b_2$ among solution that fulfil the constraint. Moreover, if there already is a solution, $x_{prev}$, in the same box as $x'$, it will be replaced by $x'$ because $Cost(x_{prev}) + 2 \cdot LP(x_{prev}) > 2 \cdot OPT$; otherwise, it would have been selected as $x$.

There are at most $A = 1 + \lceil \frac{\log n + \log W_{max}}{\log(1+\delta)} \rceil$ different values for $b_2$ in the objective space, and since $\delta = \frac{1}{2n}$, $A = O(n \cdot (\log n + \log W_{max}))$. Therefore, the expected time until a solution $x''$ is found so that $b_2(x'') = 0$ and $Cost(x'') + 2 \cdot LP(x'') \leq 2 \cdot OPT$, is at most $O(n^3 \cdot (\log n + \log W_{max})^2)$. □

## 7.5 Conclusion

The minimum vertex cover problem is one of the classical NP-hard combinatorial optimization problems. In this chapter, we have generalized previous results of Kratsch and Neumann [77] for the unweighted minimum vertex cover problem to the weighted case where in addition weights on the nodes are given. We have proved upper bounds for the expected optimization time of Global SEMO with standard mutation operator; showing that this algorithm efficiently computes a 2-approximation as long as the value of an optimal solution is small. Furthermore, we have studied the algorithm DEMO using the $\varepsilon$-dominance approach and proved that the population size is polynomially bounded with this technique. Consequently, we proved that this algorithm reaches a 2-approximation in expected polynomial time.

# Chapter 8

# Maintaining 2-Approximations for the Dynamic Vertex Cover Problem

## 8.1 Introduction

In this chapter we contribute to the theoretical understanding of evolutionary algorithms for the classical vertex cover problem which is defined in Section 3.3. Several algorithms are known for this problem that have approximation ratio of 2. The goal of our investigations in this chapter is to contribute to the understanding of how evolutionary algorithms can maintain a 2-approximation when dynamic changes such as edge addition and deletion are applied to the current graph.

Evolutionary algorithms in the dynamic optimization have previously been theoretically analysed in a number of papers for a simple (1+1) EA and the OneMax problem or a variation of it [38, 73, 108]. The paper by Droste [38] presents an interesting analysis for dynamically changing problems, where the maximum degree of dynamic changes is found such that the expected optimization time of (1+1) EA is still polynomial for the studied problem. One bit of the objective bit-string in that paper changes at each timestep with a probability $p'$; which results in the dynamic changes of the fitness function over time. The author of the paper has proved that the (1+1) EA has a polynomial expected runtime if $p' = O(log(n)/n)$, while for every substantially larger probability the runtime becomes super polynomial. The results of that paper hold even if the expected re-optimization time of the problem is larger than the expected time until the next dynamic change happens. In our analysis, we are considering a simpler dynamic setting where the rate of dynamic changes is small enough, to re-optimize the problem after a dynamic change, before the following change happens.

We study different variants of the classical randomised local search (RLS) and (1+1) EA that have already been investigated for the static vertex cover problem in the context of approximations. This includes a node-based representation examined in [48, 77, 95] as well as different edge-based representations analysed in [70].

For both of the representations there are hard instances introduced [48, 70] in which with high probability a 2-approximation solution can not be found in less than exponential time by means of (1+1) EA. Nevertheless, inspired by the approximation algorithms for the vertex cover problem using maximal matchings, Jansen et al. [70] have suggested that evolutionary algorithms using edge-based representation instead of the node-based representation can solve the problem faster. They have studied the node-based approach in addition to two edge-based approaches which differ in the fitness function, and have shown that using the edge-based representation with a fitness function that penalizes the edges that share nodes, the algorithm can find a 2-approximate solution in $O(m \log m)$ where $m$ denotes the number of edges in the given graph.

We adapt the three approaches of Jansen et al. [70] to the dynamic vertex cover problem [67] where edges may be added or deleted from the graph. For the first two approaches, we point out where they are not able to maintain 2-approximations for the dynamic vertex cover problem. In contrast to this, we show that the third approach maintains solutions of that quality very efficiently.

This chapter is based on a GECCO conference paper ([100]) and is structured as follows. In Section 8.2 the problem definition is given and the algorithms are introduced. Section 8.3 and 8.4 include hard instances of the dynamic vertex cover problem for the node-based and edge-based approaches respectively. Run time behaviour of the edge-based approach with the complex fitness function is analysed in Section 8.5 and the conclusion is presented in the last section.

## 8.2 Algorithms and the Dynamic Vertex Cover Problem

In dynamic version of vertex cover problem, the given instance is subject to the addition and deletion of edges. We assume that these changes happen one by one each $\tau$ iterations where $\tau \in poly(n)$ and $poly(n)$ is a polynomial function in $n$.

In most of the work on the vertex cover problem using evolutionary algorithms, the natural node-based representation is used [48, 77, 95]. In this representation the search space is $\{0,1\}^n$ where $n$ is the number of nodes in the graph. A potential solution is a search point $s \in \{0,1\}^n$ describing a selection of nodes, i. e. the 1-bits identify the nodes

**Algorithm 18:** Node-Based RLS (RLS$_{NB}$)

1: The initial solution, $s$, is given: a bit-string of size $n$ which used to be a 2-approximate solution before changing the graph.
2: Set $s' = s$
3: Select $i \in \{1, \ldots, n\}$ uniformly at random and flip $i$th bit of $s'$
4: If $f(s') \leq f(s)$ then $s := s'$
5: If stopping criteria not met continue at line 2

**Algorithm 19:** Node-Based (1+1) EA ((1+1) EA$_{NB}$)

1: The initial solution, $s$, is given: a bit-string of size $n$ which used to be a 2-approximate solution before changing the graph.
2: Set $s' = s$
3: Flip each bit of $s'$ with probability $\frac{1}{n}$
4: If $f(s') \leq f(s)$ then $s := s'$
5: If stopping criteria not met continue at line 2

that are in the cover-set for that solution:

$$V_C(s) = \{v_i \in V \mid s_i = 1\}.$$

In the work of Jansen et al. [70] the edge-based representation is introduced for this problem. In this representation the search space is $\{0,1\}^m$ where $m$ is the number of edges in the graph, and a search point $s \in \{0,1\}^m$ describes a selection of edges $E(s) = \{e_i \in E \mid s_i = 1\}$. The cover set then is the subset of all vertices that are on either side of the selected edges:

$$V_C(s) = \{v \in V \mid \exists e \in E(s) : v \in e\}.$$

Note that in the dynamic version of the problem, the size of the bit-string corresponding to a search point increases and decreases when edges are added or removed respectively. In our analysis, $m$ is the largest number of edges in the graph at all stages.

Jansen et al. [70] have suggested that this representation can help evolutionary algorithms solve the problem faster. They first investigated the fitness function

$$f(s) = |V_C(s)| + (|V| + 1) \cdot |\{e \in E \mid e \cap V_C(s) = \emptyset\}|. \tag{8.1}$$

The first part of this fitness function is the cardinality of the cover set which needs to be minimised. The second part is a penalty for the edges that this set does not cover.

For $f(s)$ of Equation 8.1, they have managed to show that a (1+1) EA performs equally poor in worst cases for both defined representations. Furthermore, Jansen et al. [70]

**Algorithm 20:** Edge-Based RLS (RLS$_{EB}$)

1: The initial solution, $s$, is given: a bit-string of size $m$ which used to be a 2-approximate solution before changing the graph.
2: Set $s' = s$
3: Select $i \in \{1, \ldots, m\}$ uniformly at random and flip $i$th bit of $s'$
4: If $f(s') \leq f(s)$ then $s := s'$
5: If stopping criteria not met continue at line 2

**Algorithm 21:** Edge-Based (1+1) EA ((1+1) EA$_{EB}$)

1: The initial solution, $s$, is given: a bit-string of size $m$ which used to be a 2-approximate solution before changing the graph.
2: Set $s' = s$
3: Flip each bit of $s'$ with probability $\frac{1}{m}$
4: If $f(s') \leq f(s)$ then $s := s'$
5: If stopping criteria not met continue at line 2

have shown that with adding a penalty for adjacent edges to the fitness function, the (1+1) EA with edge-based representation can solve the vertex cover problem in $O(m \log m)$. The fitness function with an extra penalty that they have used is defined as

$$\begin{aligned} f_e(s) &= f(s) + (|V| + 1) \cdot (m + 1) \\ &\quad \cdot |\{(e, e') \in E(s) \times E(s) \mid e \neq e', e \cap e' \neq \emptyset\}|. \end{aligned} \tag{8.2}$$

The fitness function $f_e(s)$ is inspired by the well-known approximation algorithm that finds a 2-approximation for the vertex cover problem based on a maximal matching [17].

In this chapter, we analyse the behaviour of RLS and the (1+1) EA on the dynamic vertex cover problem with similar approaches that were studied in [70] on vertex cover problem. These algorithms are supposed to modify the given solution if needed to make it keep its quality of 2-approximation, after an edge has been added to or deleted from the graph. In our runtime analysis, we measure runtime by the number of fitness evaluations to reach a certain goal. The expected runtime refers to the expected number of fitness evaluations to reach the desired goal. In our case, the goal is to recompute a 2-approximation after one or more sequentially applied dynamic changes have occurred.

The Node-Based RLS (RLS$_{NB}$) and Node-Based (1+1) EA ((1+1) EA$_{NB}$) are defined in Algorithm 18 and Algorithm 19, respectively. Similarly, Edge-Based RLS (RLS$_{EB}$) and Edge-Based (1+1) EA ((1+1) EA$_{EB}$) using the fitness function $f(s)$ are presented in Algorithm 20 and 21, respectively. The definition of the algorithms for the third approach is quite similar to the second approach except that for comparing two solutions in line

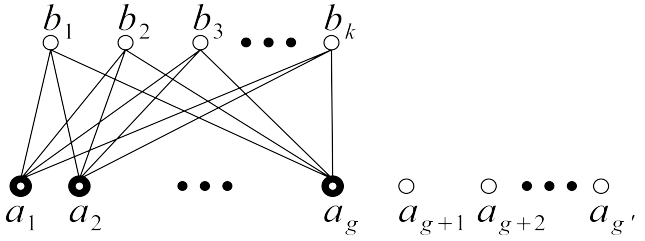


FIGURE 8.1: $G_1$, a hard instance for node-based approach

4, the fitness function with the extra penalty of Formula 8.2 is used instead of the simple fitness function $f(s)$. In the following sections, we denote the RLS and (1+1) EA variants of the third approach by RLS$_e$ and (1+1) EA$_e$, respectively.

## 8.3 Hard Instance for Node-Based Approach

In this section, we introduce a bipartite graph for which it is hard to maintain a 2-approximate solution by means of RLS$_{NB}$ and (1+1) EA$_{NB}$. We go even further by showing that for our instance and a sequence of dynamic changes, only a very bad approximation will be found by these two algorithms. In our instance, both of the algorithms stick to a local optimum with a bad approximation ratio of $\Omega(n^{1-\epsilon})$, $\epsilon > 0$ a small constant, if the graph is subject to a polynomial number of changes. In this section we assume that $\tau \geq n^{(3+\delta)}$ and $\delta > 0$ is a small constant. An Illustration of our instance, $G_1$, is given in Figure 8.1.

$G_1 = (V, E)$ is a bipartite graph and the set of nodes, $V$, is partitioned into two sets $W = \{a_1, \ldots, a_{g'}\}$ and $U = \{b_1, \ldots, b_k\}$. If $n$ denotes the total number of nodes, then we assume that $k = \frac{1}{3}n^{\epsilon}$ and $g' = n - \frac{1}{3}n^{\epsilon}$. Initially, $g = 2k$ nodes from part $W$ are connected to all the nodes of part $U$, i. e. the sub-graph consisting of nodes $U \cup \{a_1, \ldots, a_g\}$ and all edges between them, which we denote by ${G_1}'$, is a complete bipartite graph. The other nodes, $\{a_i \mid g+1 \leq i \leq g'\}$, are initially not adjacent to any edge, but will be connected to the nodes of part $U$ one by one. In other words, the dynamic changes that the graph is subject to, are adding edges $\{\{a_i, b_j\} \mid g+1 \leq i \leq g', 1 \leq j \leq k\}$. Among these, edges $\{e \mid a_i \in e\}$ are added prior to edges $\{e \mid a_{i+1} \in e\}$; and edge $\{a_i, b_j\}$, is added prior to edge $\{a_i, b_{j+1}\}$. The number of these edges is $(g' - g) \cdot k = (n - n^{\epsilon}) \cdot (\frac{1}{3}n^{\epsilon}) = O(n^{1+\epsilon})$.

*Property* 8.1. The optimal solution for $G_1$ at all stages of dynamic changes, is the set $U$ of size $k$.

*Proof.* At all stages, the sub-graph $G_1'$ is a complete bipartite graph; and a vertex cover for a complete bipartite graph, contains at least all of the nodes of one of the parts.

Therefore, either all the nodes of $U$ need to be in the solution or all the nodes $\{a_i \mid 1 \leq i \leq g\}$. Since $g = 2k$, any cover set containing $\{a_i \mid 1 \leq i \leq g\}$, has a size of at least $2k$; whereas any cover set containing $U$ has a size of at least $k$. Therefore, the optimal solution has a size of at least $k$.

On the other hand, the set $U$ is a complete cover for $G_1$ at all the stages because it is one of the partite sets of $G_1$. Therefore, the optimal cover set at all the stages of the graph is $U$ with a size of $k$. □

The initial 2-approximate solution that is given consists of $\{a_i \mid 1 \leq i \leq g\}$. This is a 2-approximate solution because $g = 2k$ and the optimal solution has a size of $k$ as we saw in Property 8.1. In Sections 8.3.1 and 8.3.2 we show that algorithms RLS$_{NB}$ and (1+1) EA$_{NB}$ find a locally optimal solution consisting of all nodes of $W$ when the dynamic changes are done and $G_1$ is a complete bipartite graph.

Based on the fitness function $f(s)$ that is used in the node-based approach, we bring 6 lemmata here that hold for both RLS$_{NB}$ and (1+1) EA$_{NB}$, and help us with the proofs in Section 8.3.1, and 8.3.2.

**Lemma 8.2.** *If the number of uncovered edges by the current solution $s$ is $A$, then any solution $s'$ with $B$ uncovered edges is rejected by RLS$_{NB}$ and (1+1) EA$_{NB}$ if $B > A$.*

*Proof.* Recalling Equation 8.1, $f(s) = |V_C(s)|+(n+1)\cdot A$. Since $B > A$, $f(s') \geq |V_C(s')|+(n+1)\cdot(A+1)$. Moreover, the maximum and minimum value of $|V_C(s)|$ and $|V_C(s')|$ are $n$ and 0 respectively. As a result $f(s) \leq n+(n+1)\cdot A$ and $f(s') \geq 0+(n+1)\cdot(A+1)$. Since $(n+1)\cdot(A+1) > (n+1)\cdot A$, this upper and lower bounds on $f(s)$ and $f(s')$ imply that $f(s') > f(s)$ and $s'$ will be rejected by RLS$_{NB}$ and (1+1) EA$_{NB}$. □

**Lemma 8.3.** *If the current solution $s$, is a cover, any solution $s'$ where $|V_C(s')| > |V_C(s)|$, is rejected by RLS$_{NB}$ and (1+1) EA$_{NB}$.*

*Proof.* For any solution $s'$ the following inequality holds:

$$f(s') \geq |V_C(s')|$$

Since solution $s$ is a cover, we have $f(s) = |V_C(s)|$. Therefore, $|V_C(s')| > |V_C(s)|$ implies that $f(s') > f(s)$, which results in rejecting $s'$ by RLS$_{NB}$ and (1+1) EA$_{NB}$. □

**Lemma 8.4.** *If the solution $s$ is a cover, with probability $1-e^{-\Omega(n^\epsilon)}$, RLS$_{NB}$ and (1+1) EA$_{NB}$ find a solution $s'$ which is a minimal cover, in time $O(n^{1+\epsilon}\log n)$.*

*Proof.* According to Lemma 8.2, any solution that is accepted by RLS$_{NB}$ or (1+1) EA$_{NB}$ after solution $s$, is a cover. Assume $s'$ to be a cover with $|V_C(s')| < |V_C(s)|$. According to Lemma 8.3, $s'$ is better in terms of fitness and will replace solution $s$. Since $s$ is not a minimal cover, there are some extra nodes in it, removing which does not uncover any edges while reduces the size of the cover set. The process of removing extra nodes from the solution is similar to optimizing OneMax [70] and is expected to be done in time $O(n \log n)$ by RLS and (1+1) EA since $|V_C(s)| \leq n$.

If the expected time until all of extra nodes are removed is $Cn \log n$, where $C$ is a constant, and if $X$ is the first time that they are removed from the solution, then by Markov's inequality (Section 4.2.1) we have

$$Prob(X \geq 2Cn \log n) \leq \frac{1}{2}$$

Considering $n^\epsilon$ phases of $2Cn \log n$ steps, then

$$Prob(X \geq 2Cn^{1+\epsilon} \log n) \leq \left(\frac{1}{2}\right)^{n^\epsilon}$$

As a result, with probability $1-e^{-\Omega(n^\epsilon)}$ a minimal cover will be found in time $O(n^{1+\epsilon} \log n)$. □

**Lemma 8.5.** *If the solution $s$ is a cover before the new edge $e$ is added, then with probability $1 - e^{-\Omega(n^\epsilon)}$ starting with $s$, (1+1) EA$_{NB}$ and RLS$_{NB}$ find a solution $s'$ which is also a cover after $e$ is added, in time $O(n^{1+\epsilon})$.*

*Proof.* Since $s$ had been a cover before $e$ was added, the only edge that might not be covered by $s$ is $e$. Therefore, the number of uncovered edges is at most 1. This number does not increase according to Lemma 8.2 during the process of RLS$_{NB}$ and (1+1) EA$_{NB}$. On the other hand, there are always two nodes (included in the uncovered edge itself) that adding at least one of them to the solution $s$, results in a cover. This move is accepted according to Lemma 8.2 and has the probability of $\Omega(n^{-1})$ for both RLS$_{NB}$ and (1+1) EA$_{NB}$. Therefore, the expected time until this improvement is found is $Cn$, $C$ a positive constant, and using Markov's inequality and $n^\epsilon$ phases of $2 \cdot Cn$ steps (similar to proof of Lemma 8.4), with probability $1 - e^{-\Omega(n^\epsilon)}$, a cover will be found in time $O(n^{1+\epsilon})$. □

**Lemma 8.6.** *Consider the given solution $s$, a cover which does not include $b_{j'}$; $3 \leq j' \leq k$, before new edge $e = \{a_i, b_j\}$ is added to the graph. With probability $1 - e^{-\Omega(n^\epsilon)}$, the resulting solution of (1+1) EA$_{NB}$ and RLS$_{NB}$ after $e$ is added, includes at least all nodes $a_{i'}$; $1 \leq i' \leq i - 1$.*

*Proof.* As $s$ does not include $b_{j'};\ 3 \leq j' \leq k$, it must contain all $a_{i'};\ 1 \leq i' \leq i-1$; otherwise, it is not a cover before $e = \{a_i, b_j\}$ is added. Therefore, the only edge that might not be covered after $e$ is added, is $e$ itself. The number of uncovered edges does not increase during the process of (1+1) $\text{EA}_{NB}$ and $\text{RLS}_{NB}$ (Lemma 8.2); therefore, none of $a$-nodes can be removed from the solution unless all $b$-nodes are added at the same step. This move is not possible with $\text{RLS}_{NB}$ because only single-bit flips can be done in that algorithm. (1+1) $\text{EA}_{NB}$ also needs to flip at least $k-2$ bits at one step which has the probability of at most $e^{-\Omega(k)} = e^{-\Omega(n^\epsilon)}$. As a result, starting from $s$, with probability $1 - e^{-\Omega(n^\epsilon)}$, any solution accepted by (1+1) $\text{EA}_{NB}$ and $\text{RLS}_{NB}$ after $e$ is added, includes at least all nodes $a_{i'};\ 1 \leq i' \leq i-1$. □

**Lemma 8.7.** *Consider $g < i \leq g'$ and two stages of the graph $G_1$: stage $X$ at which $\{a_i, b_2\}$ is added, and stage $Y$ which is before $\{a_{i+1}, b_1\}$ is added. At all stages from $X$ to $Y$, a solution consisting of $\{a_1, \ldots, a_i\}$ is a locally optimal solution for $\text{RLS}_{NB}$. Moreover, with probability of $1 - e^{-\Omega(n^\epsilon)}$, (1+1) $\text{EA}_{NB}$ can not improve this solution in a polynomial number of steps.*

*Proof.* Since edges connected to $a_{i'};\ i' > i$ have not been added to the graph yet, $s$, consisting of $\{a_1, \ldots, a_i\}$, is a cover. Therefore, any solution that has at least one uncovered edge (Lemma 8.2) or has a larger number of nodes in the cover set(Lemma 8.3) is rejected by $\text{RLS}_{NB}$ and (1+1) $\text{EA}_{NB}$.

Since the sub-graph consisting of the nodes $\{a_1, \ldots, a_i\} \cup U$ and all edges between them, is a complete bipartite graph, any solution which is a cover, must contain either $U$ or $\{a_1, \ldots, a_i\}$. Similar to Lemma 8.6, jumping to any solution which contains $U$, in one step by $\text{RLS}_{NB}$ is not possible and by (1+1) $\text{EA}_{NB}$ has a probability of $e^{-\Omega(n^\epsilon)}$. Moreover, among solutions containing the set $\{a_1, \ldots, a_i\}$, $s$ has the minimum number of nodes in the cover set. Therefore, $s$ is a local optimum for $\text{RLS}_{NB}$ and with probability $e^{-\Omega(n^\epsilon)}$, (1+1) $\text{EA}_{NB}$ can not improve it in a polynomial number of steps. □

### 8.3.1 Analysis of $\text{RLS}_{NB}$ on $G_1$

In this section, we first bring two lemmata that help us identify local optimums. Using them, we analyse the behaviour of $\text{RLS}_{NB}$ on $G_1$.

**Lemma 8.8.** *Any solution $s$ which is a minimal cover, is a locally optimal solution for $\text{RLS}_{NB}$.*

*Proof.* Only single-bit flips can be preformed by $\text{RLS}_{NB}$ which can be either adding a new node to $s$ or removing one from it. Since $s$ is a cover, adding new nodes to it is rejected according to Lemma 8.3 because it increases $|V_C(s)|$. Moreover, any move that

removes a node, uncovers at least one edge as $s$ is a minimal cover. Therefore, according to Lemma 8.2 deleting nodes from $s$ also is rejected; hence, $s$ is a local optimum. □

**Lemma 8.9.** *Consider $g < i \leq g'$ and two stages of the graph $G_1$: stage $X$ at which $\{a_i, b_1\}$ is added, and stage $Y$ which is before $\{a_{i+1}, b_1\}$ is added. If the given solution of RLS$_{NB}$ at stage $X$ is $\{a_1, \ldots, a_{i-1}\}$, then with probability $1 - e^{-\Omega(n^\epsilon)}$, the resulting solution of the algorithm at stage $Y$ is $\{a_1, \ldots, a_i\}$.*

*Proof.* After stage X, when the edge $\{a_i, b_1\}$ is added to the graph, the current solution, $\{a_1, a_2, \ldots, a_{i-1}\}$, is not a cover any more.

The RLS$_{NB}$ flips bits of search point $s$, one bit at a time, to achieve an improvement on the fitness $f(s)$. Flipping a 1 to 0 indicates removing a node from the solution; which uncovers $k$ edges and according to Lemma 8.2, is rejected. Flipping a 0 to 1 increases $|V_C(s)|$, but if the corresponding node covers the new edge; then the fitness is improved by $n$. If not, the fitness is increased by 1. Therefore, the only flips that are accepted by RLS$_{NB}$ are the ones that add a node that covers the new edge. The only such nodes are $a_i$ and $b_1$. Note that in both cases the resulting solution is worse than a 2-approximation, because it contains $i \geq 2k + 1$ nodes.

At each step, the probability of selecting one of these two nodes is $\frac{2}{n}$. Therefore, the expected time until one of them is selected is $\frac{n}{2}$. Let $X$ be the first time that one of them is selected. Using Markov's inequality and $n^\epsilon$ phases of $n$ steps (similar to proof of Lemma 8.4) with probability $1 - e^{-\Omega(n^\epsilon)}$, the solution is improved in $n^{1+\epsilon}$ steps.

If $a_i$ is added, then according to Lemma 8.7 the solution is a local optimum until edge $\{a_{i+1}, b_1\}$ is added. Here we consider adding $b_1$ to the cover set.

At this stage, i.e. before the next edge is added to the graph, the solution $\{a_1, a_2, \ldots, a_{i-1}, b_1\}$, is a minimal cover and a local optimum according to Lemma 8.8.

Similar to the first one, when the edge $\{b_2, a_i\}$ is added to the graph, the current solution is no more a complete cover. Either $b_2$ or $a_i$ need to be added to the cover set. The situation is similar to what we had for the first change and will be repeated in the next stages while the $b$-node is selected.

For each of the $k$ edges $\{a_i, b_j\}; 1 \leq j \leq k$, the probability that the $b$-node is added to the cover set instead of $a_i$ is $\frac{1}{2}$. Therefore, with probability $1 - (\frac{1}{2})^k = 1 - e^{-\Omega(n^\epsilon)}$, at least one of these changes results in adding $a_i$ to the selected set of nodes. Let us assume that the $\{a_i, b_j\}$ is the first edge for which $a_i$ is added to the cover set. The new solution $\{a_1, \ldots, a_i, b_1, \ldots, b_{j-1}\}$ is a cover, but not a minimal cover, because removing $b$-nodes from the set does not uncover any edges. According to Lemma 8.4, they will be

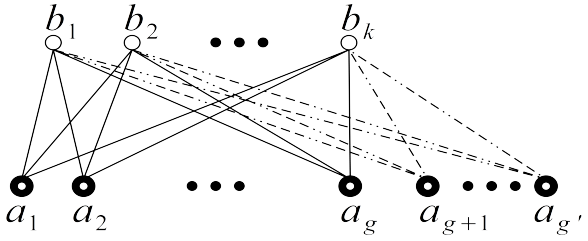


FIGURE 8.2: A solution consisting of set $W$

removed from the solution and with probability $1-e^{-\Omega(n^\epsilon)}$ the locally optimal solution of $\{a_1,\ldots,a_i\}$ (Lemma 8.7) will be found in time $O(n^{1+\epsilon}\log n)$. □

**Theorem 8.10.** *For $G_1$, with probability $1-e^{-\Omega(n^\epsilon)}$, the eventual resulting solution of RLS$_{NB}$ is the set $W$. The approximation ratio of this solution is in $\Omega(n^{1-\epsilon})$.*

*Proof.* Since the initial solution is $\{a_1,\ldots,a_g\}$, according to Lemma 8.9, the resulting solution of RLS$_{NB}$ before edge $\{a_{g+2},b_1\}$ is added is $\{a_1,\ldots,a_{g+1}\}$, with probability $1-e^{-\Omega(n^\epsilon)}$. This satisfies the requirement for Lemma 8.9 again, and the algorithm repeats the whole process for each of $(n-n^\epsilon)$ $a$-nodes. As a result, after node is added, the algorithm finds a solution consisting of $W$ with probability $1-(n-n^\epsilon)e^{-\Omega(n^\epsilon)} = 1-e^{-\Omega(n^\epsilon)}$ (Figure 8.2).

There are $g=n-\frac{1}{3}n^\epsilon$ nodes in this solution, which gives the approximation ratio of:

$$\frac{g}{k}=\frac{n-\frac{1}{3}n^\epsilon}{\frac{1}{3}n^\epsilon}=\Omega(n^{1-\epsilon})$$

□

### 8.3.2 Analysis of (1+1) EA on $G_1$

We here introduce a lemma using which the main theorem of this section about the behaviour of (1+1) EA$_{NB}$ on $G_1$ is proved.

**Lemma 8.11.** *Consider $g<i\leq g'$ and two stages of the graph $G_1$: stage $X$ at which $\{a_i,b_1\}$ is added, and stage $Y$ which is before $\{a_{i+1},b_1\}$ is added. If the given solution of (1+1) EA$_{NB}$ at stage $X$ is $s=\{a_1,\ldots,a_{i-1}\}$, then with probability $1-e^{-\Omega(n^\delta)}$, the resulting solution of the algorithm at stage $Y$ is $s'=\{a_1,\ldots,a_i\}$.*

*Proof.* The given solution $\{a_1,\ldots,a_{i-1}\}$ is a cover for the graph before stage $X$. According to Lemma 8.5, after $\{a_i,b_1\}$ is added, the algorithm finds $s_1$ which is a cover,

in polynomial time. Similarly, after $\{a_i, b_2\}$ is added, a solution $s_2$ that is also a cover is found in polynomial time.

On the other hand, according to Lemma 8.6, $s_1$ contains all nodes $a_{i'};\ 1 \leq i' \leq i-1$. And according to Lemma 8.4, it is a minimal cover because $\tau > O(n^{1+\epsilon}\log n)$, meaning that in addition to $\{a_1, \ldots, a_{i-1}\}$, it only includes one other node to cover $\{a_i, b_1\}$ i.e. either $a_i$ or $b_1$. Therefore, $s_1$ does not include $b_{j'};\ 3 \leq j' \leq k$; which satisfies the condition of Lemma 8.6 for the next stage. In other words, $s_2$ also includes all nodes $a_{i'};\ 1 \leq i' \leq i-1$. According to Lemma 8.4 it is also a minimal cover. Any minimal solution including all nodes $a_{i'};\ 1 \leq i' \leq i-1$ which also covers $\{a_i, b_1\}$ and $\{a_i, b_2\}$, contains either $a_i$ or $b_1$ and $b_2$. So far, we have proved that (1+1) EA$_{NB}$ finds $s_2$ which consists of either $V_1 = \{a_1, \ldots, a_i\}$ or $V_2 = \{a_1, \ldots, a_{i-1}, b_1, b_2\}$. We here show that (1+1) EA$_{NB}$ finds the solution which includes $V_1$.

Since both $V_1$ and $V_2$ are covers and $|V_1| < |V_2|$, the solution consisting of $V_1$ is less costly. Therefore, (1+1) EA$_{NB}$ does not accept a change from $V_1$ to $V_2$ but accepts the opposite move. The probability of flipping only the bits corresponding to $a_i$, $b_1$ and $b_2$ is

$$\left(\frac{1}{n}\right)^3\left(1-\frac{1}{n}\right)^{n-3} \geq \frac{1}{en^3}$$

As a result, a move from $V_2$ to $V_1$ can be performed at expected time of at most $E(T) = en^3$, where $T$ is the first step that this move happens. Using Markov's inequality (Section 4.2.1),

$$Prob(T \geq 2 \cdot en^3) \leq \frac{1}{2}$$

Considering $\frac{n^{(3+\delta)}}{2en^3}$ phases of $2en^3$ steps,

$$Prob(T \geq n^{3+\delta}) \leq \left(\frac{1}{2}\right)^{\frac{n^{(3+\delta)}}{2en^3}}$$

This implies that with probability $1 - e^{-\Omega(n^\delta)}$ the (1+1) EA$_{NB}$ finds $s_2$ consisting of $V_1$ in a phase of $\tau$ steps.

From this point according to Lemma 8.7, $s_2$ is a local optimum until $\{a_{i+1}, b_1\}$ is added. □

**Theorem 8.12.** *For $G_1$, with probability $1-e^{-\Omega(n^\delta)}$, the eventual resulting solution of (1+1) EA$_{NB}$ is the set $W$. The approximation ratio of this solution is in $\Omega(n^{1-\epsilon})$.*

*Proof.* Since the initial solution is $\{a_1, \ldots, a_g\}$, according to Lemma 8.11, the resulting solution of (1+1) EA$_{NB}$ before edge $\{a_{g+2}, b_1\}$ is added is $\{a_1, \ldots, a_{g+1}\}$, with probability $1 - e^{-\Omega(n^\delta)}$. This satisfies the requirement for Lemma 8.11 again, and the algorithm

repeats the whole process for each node $a_i$, $g < i \leq g'$. As a result, after the last change of the graph, the algorithm finds a solution consisting of all nodes of $W$ with a probability of at least $1 - (n - n^{\epsilon})e^{-\Omega(n^{\delta})} = 1 - e^{-\Omega(n^{\delta})}$.

Similar to Theorem 8.10 the approximation ratio of this solution is $\Omega(n^{1-\epsilon})$. □

## 8.4 Hard Instance for Standard Edge-Based Approach

In this section, we assume that $\tau \geq m^{(4+\delta)}$, where $\delta > 0$ is a small constant. The graph of instance $G_2$ that we introduce in this section as a hard instance for edge-based approach is exactly the same as $G_1$ with one slight difference. The difference is that $g = 2k - 1$ instead of $2k$.

*Property* 8.13. The optimal solution for $G_2$ at all stages of dynamic changes, is the set $U$ of size $k$.

*Proof.* The proof is similar to the proof of Property 8.1 on $G_1$. □

Since we are using the edge-based representation in this section, the initial solution needs to be a search point in $\{0,1\}^m$, representing the set of selected edges. We assume that $\{\{a_i, b_1\} \mid 1 \leq i \leq g\}$ is the given initial set of selected edges. The cover set induced from this set is $\{b_1\} \cup \{a_i \mid 1 \leq i \leq g\}$ which has a size of $2k$; therefore, is a 2-approximation because according to Property 8.13 the size of the optimal solution is $k$. In what follows, we analyse the behaviour of RLS$_{EB}$ and (1+1) EA$_{EB}$ on $G_2$ with the given initial solution.

Note that Lemma 8.2 and Lemma 8.3 of Section 8.3 are also valid in this section, because they are based on the fitness function $f(s)$ and not the representation. We bring three other lemmata here which hold for both RLS$_{EB}$ and (1+1) EA$_{EB}$.

**Lemma 8.14.** *Starting with the given solution $s$ which is a cover before edge $e$ is added, with probability $1 - e^{-\Omega(m^{\epsilon})}$, RLS$_{EB}$ and (1+1) EA$_{EB}$ find a solution $s'$ which is also a cover in $O(n^{1+\epsilon})$ after $e$ is added.*

*Proof.* The proof of this lemma is similar to the proof of Lemma 8.5 except that we are using the edge-based approach and we should use the probability of finding the proper edge instead of the probability of finding the proper node.

There always exists an edge (the uncovered edge itself) adding which to $s$ results in a cover. This move has the probability of $\Omega(m^{-1})$ for both RLS$_{EB}$ and (1+1) EA$_{EB}$.

Therefore, similar to Lemma 8.5, with probability $1 - e^{-\Omega(m^\epsilon)}$, a cover will be found in $O(m^{1+\epsilon})$. □

**Lemma 8.15.** *For the instance $G_2$ starting with the given initial solution, with probability $1 - e^{-\Omega(m^\epsilon)}$, RLS$_{EB}$ and (1+1) EA$_{EB}$ result in a solution which is a cover at all stages.*

*Proof.* Using Lemma 8.14 as inductive steps and the initial solution as the base of induction, we can conclude that with probability $1 - e^{-\Omega(m^\epsilon)}$, RLS$_{EB}$ and (1+1) EA$_{EB}$ result in a solution which is a cover at all stages. □

**Lemma 8.16.** *Before new edge $e = \{a_i, b_j\}$ is added to the graph, consider the given solution $s$, a cover which does not include $b_{j'};\ 4 \leq j' \leq k$. With probability $1 - e^{-\Omega(m^\epsilon)}$, the resulting solution of (1+1) EA$_{EB}$ and RLS$_{EB}$ after $e$ is added, includes at least all nodes $a_{i'};\ 1 \leq i' \leq i - 1$.*

*Proof.* The proof is similar to proof of Lemma 8.6. However, notice that with edge-based representation, adding and removing nodes from the cover set of solution $s$ can be done by adding and removing edges from $s$. Since $G_2$ is a bipartite graph, adding all nodes $b_{j'};\ 4 \leq j' \leq k$ at one step, requires at least $k = 3$ flips. □

### 8.4.1 Analysis of RLS$_{EB}$ on $G_2$

In this section, we analyse the behaviour of RLS$_{EB}$ on $G_2$ and show that there is a stage at which this algorithm fails to find a solution with a better approximation ratio than $\frac{3k-1}{k}$. We first bring a lemma that helps us with the proof.

**Lemma 8.17.** *With probability $1 - e^{-\Omega(m^\epsilon)}$, the node $b_k$ can only be added to the solution by RLS$_{EB}$, at a stage in which $\{a_i, b_k\}$ has been added to the graph, where $g + 1 \leq i \leq g'$.*

*Proof.* Consider stage $X$ in which node $b_k$ is added to the solution. If adding an edge $e = \{a_{i'}, b_k\}; 1 \leq i' \leq g'$ is accepted, it must cover an uncovered edge; otherwise, this move will be rejected because it increases the cardinality of the cover set. On the other hand, according to Lemma 8.15, the solution obtained by the algorithm before stage $X$, is a cover; therefore the only uncovered edge is the one that is added in stage $X$ which we denote by $e_n$. As a result, $e$ is covering $e_n$; i.e. $e_n = \{a_i, b_k\};\ g + 1 \leq i \leq g'$ which is what the lemma claims, or, $e_n = \{a_{i'}, b_{k'}\};\ 1 \leq k' \leq k$. In this case, $k' = k$ because otherwise, the edge $\{a_{i'}, b_k\}$ has not been added to the graph yet. This completes the proof. □

**Theorem 8.18.** *Before edge $\{a_{3k}, b_1\}$ is added to $G_2$, with probability $1 - e^{-\Omega(m^\epsilon)}$, there is a stage at which RLS$_{EB}$ does not find any solution better than $\frac{3k-1}{k}$-approximation.*

*Proof.* Before $\{a_{3k}, b_1\}$ is added to the graph, all edges connected to nodes $a_i;\ 2k \leq i < 3k$, are added. We partition these stages into $k$ phases so that in phase $i$ ($2k \leq i < 3k$), edges $\{a_i, b_j \mid 1 \leq j \leq k\}$ are added.

We analyse the situation based on containing or not containing the node $b_k$ in the obtained solution of at least one stage.

- If an edge containing nodes $b_k$ is added to the solution, according to Lemma 8.17, it must have been added after edge $e_n = \{a_i, b_k\};\ 2k \leq i < 3k$ is added to $G_2$. Furthermore, adding $b_k$ to a solution $s$ increases $|V_C(s)|$. If $s$ was a cover, according to Lemma 8.3 this move was rejected. Therefore, $b_k$ must have covered a new edge. According to Lemma 8.15, before $e_n$ is added; the algorithm has managed to find a cover; therefore, the only edge that may not be covered after adding $e_n$, is $e_n$ itself. Therefore, the edge containing $b_k$, that is added to the solution, must cover $e_n = \{a_i, b_k\}$, to be accepted. This implies that the node $a_i$ has not been previously added to the solution. Otherwise, $e_n$ was already covered.

  Consider stage $X$, at which $\{a_i, b_{k-1}\}$ has been added. According to the above explanation, the solution obtained at this stage (before $e_n$ is added) must not contain $a_i$. Moreover, according to Lemma8.15, this solution is a cover. Since $a_i$ has been connected to all other $b$-nodes in the past stages, all the nodes $b_j;\ 1 \leq j \leq k-1$, must be included in that solution as it does not contain $a_i$. Similarly, all the nodes $a_l;\ 1 \leq l \leq i-1$ must be included in that solution because $b_k$ is connected to all of them and is not included in the cover set yet.

  As a result, the achieved solution by the end of stage $X$, contains nodes $a_l;\ 1 \leq l \leq i-1$ and $b_j;\ 1 \leq j \leq k-1$. Since $i \geq 2k$, the total number of nodes in the cover set is at least $3k-1$.

- If $b_k$ is not included in the solution of any stage; then the cover set must include all of the nodes that are connected to $b_k$. At the last stage of phase $3k-1$, all nodes $a_l, 1 \leq l \leq 3k-1$ are connected to $b_k$; therefore, the cover set that the algorithm finds at that stage contains at least $3k-1$ edges.

In both cases, we proved that there is a stage at which the achieved cover set contains at least $3k-1$ nodes; while the size of the optimal cover set is $k$ at all stages. Therefore, the approximation ratio of the mentioned solutions is $\frac{3k-1}{k}$. □

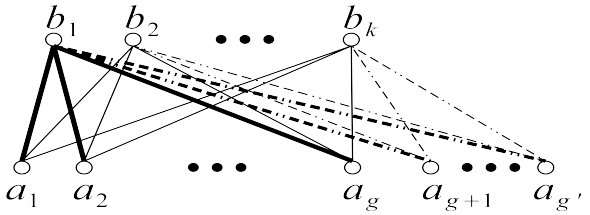


FIGURE 8.3: A solution including the set $W$

### 8.4.2 Analysis of EA on $G_2$

Here we introduce two lemmata that helps us with the proof of the main theorem about the behaviour of (1+1) $EA_{EB}$ on $G_2$. Lemma 8.19 is obtained similar to Lemma 8.7.

**Lemma 8.19.** *Consider $g < i \leq g'$ and two stages of the graph $G_1$: stage $X$ at which $\{a_i, b_3\}$ is added, and stage $Y$ which is before $\{a_{i+1}, b_1\}$ is added. At all stages from $X$ to $Y$, a solution $s$, consisting of $\{a_1, \ldots, a_i, b_j\}$; $1 \leq j \leq k$ is a locally optimal solution for (1+1) $EA_{EB}$.*

**Lemma 8.20.** *Consider $g < i \leq g'$ and two stages of the graph $G_1$: stage $X$ at which $\{a_i, b_1\}$ is added, and stage $Y$ which is before $\{a_{i+1}, b_1\}$ is added. If the given solution of (1+1) $EA_{EB}$ at stage $X$, $s$, includes the nodes $\{a_1, \ldots, a_{i-1}\}$, then with probability $1 - e^{-\Omega(m^\delta)}$, the resulting solution of the algorithm at stage $Y$ also includes $\{a_1, \ldots, a_i\}$.*

*Proof.* The given solution $s$ is a cover for the graph before stage $X$. According to Lemma 8.14, after $\{a_i, b_1\}$, $\{a_i, b_2\}$ and $\{a_i, b_3\}$ are added, the algorithm finds $s_1$, $s_2$ and $s_3$ which are also covering solutions, in polynomial time.

On the other hand, according to Lemma 8.16, $s_1$, $s_2$ and $s_3$ contain all nodes $a_{i'}$; $1 \leq i' \leq i-1$. A covering solution that contains the three new edges, must contain either $a_i$ or $b_1$, $b_2$ and $b_3$. Among solutions with these properties, $s_3 = \{\{a_l, b_j\} \mid 1 \leq l \leq i\}$ has the minimum cost and is achievable from others at each step with a probability of at least $(\frac{1}{m})^4(1-\frac{1}{m})^{(m-4)}$ because at most 4 bits of $s$ need to be flipped. Similar to proof of Lemma 8.11, we can conclude that with probability $1 - e^{-\Omega(m^\delta)}$ the (1+1) $EA_{EB}$ finds $s_3 = \{\{a_l, b_1\} \mid 1 \leq l \leq i\}$ in a phase of $\tau$ steps which is due to Lemma 8.19, a local optimum until $\{a_{i+1}, b_1\}$ is added. □

Similar to Theorem 8.12 we obtain the following result.

**Theorem 8.21.** *For $G_2$, with probability $1 - e^{-\Omega(m^\delta)}$, the (1+1) $EA_{EB}$ finds a locally optimal solution containing the set $W$ (Figure 8.3). The approximation ratio of this solution is in $\Omega(n^{1-\epsilon})$.*

## 8.5 Edge-Based Approach with extra penalty

In this section, we analyse the impact of using the fitness function $f_e(s)$, defined in Equation 8.2, on the behaviour of RLS$_e$ and (1+1) EA$_e$. It is already proved [70] that starting from any initial solution, both of these algorithms find a maximal matching in time $O(m \log m)$, which induces a 2-approximate solution for the vertex cover problem. As a result, considering $\tau \geq m^{(1+\delta)}$, both algorithms find a 2-approximate solution for the dynamic vertex cover problem with probability $1 - e^{-\Omega(\frac{m^\delta}{\log m})}$. We aim to analyse the behaviour of the two algorithms when an initial solution with that quality is given. Both kind of dynamic changes on the graph, *add* and *delete*, are analysed in this section. The following two lemmata are proved based on the fitness function $f_e(s)$ that RLS$_e$ and (1+1) EA$_e$ use.

**Lemma 8.22.** *Consider a search point $s \in \{0,1\}^m$ which is a matching. Any move that results in search point $s'$ is rejected by RLS$_e$ and (1+1) EA$_e$ if $s'$ is not a matching.*

*Proof.* We here show that for any $s$ which is a matching and any $s'$ which is not a matching, $f_e(s') > f_e(s)$; therefore, both algorithms reject $s'$.

If $s'$ is not a matching, $f_e(s') \geq (|V|+1) \cdot (m+1) = (n+1) \cdot (m+1)$. Moreover, if $s$ is a matching $f_e(s) = f(s) \leq n + (n+1)(m)$ because the maximum number of uncovered edges is $m$. On the other hand, $(n+1) \cdot (m+1) > n + (n+1)(m)$ which implies that $f_e(s') > f_e(s)$ □

**Lemma 8.23.** *Consider a search point $s \in \{0,1\}^m$ which is a matching. Any move that results in search point $s'$ is rejected by RLS$_e$ and (1+1) EA$_e$ if $|\{e \in E \mid e \cap V_C(s') = \emptyset\}| > |\{e \in E \mid e \cap V_C(s) = \emptyset\}|$.*

*Proof.* For any search point $s'$, $f_e(s') \geq f(s')$ holds. Since solution $s$ is a matching, $f_e(s) = f(s)$. Therefore, if $f(s') > f(s)$, $f_e(s') > f_e(s)$ also holds. Moreover, according to Lemma 8.2, if the number of uncovered edges of solution $s'$ is larger than that of solution $s$, $f(s') > f(s)$ holds which completes the proof. □

### 8.5.1 Analysis of RLS

In this section, using the following lemma, we prove that RLS$_e$ maintains a 2-approximate solution in $O(m)$ on expectation. This gives the probability of $1 - e^{-\Omega(m^\delta)}$ for maintaining the quality of the problem with $\tau = m^{(1+\epsilon)}$.

**Lemma 8.24.** *Any search point $s \in \{0,1\}^m$ which is a maximal matching, is a locally optimal solution for RLS$_e$.*

*Proof.* By RLS$_e$ only single-bit flips can be performed: adding one edge to $s$, or deleting one edge from it. We show that both kinds are rejected; hence $s$ is a local optimum.

Since $s$ is a matching, removing an edge from it uncovers at least one edge, and according to Lemma 8.23, is rejected. And since it is a maximal matching, adding any edge to it will result in a solution which is not a matching, and according to Lemma 8.22, is rejected. □

**Theorem 8.25.** *Starting with a 2-approximate solution $s$, which is also a maximal matching for an instance of the problem, RLS$_e$ maintains the quality of the solution for dynamic changes of adding or deleting an edge on the graph in expected time of $O(m)$.*

*Proof.* We investigate the situation for adding an edge or deleting an edge separately.

When an edge is added to the graph, $s$ is still a matching, but it might be or not be a maximal matching. If $s$ is still a maximal matching then it is a local optimum (Lemma 8.24) and a 2-approximation, because all maximal matchings induce a 2-approximation cover set.

If $s$ is not a maximal matching, then only the new edge, $e$, might not be covered. According to Lemma 8.22 and Lemma 8.23, $s$ remains matching during the process of the algorithm and the number of uncovered edges does not increase. Moreover, while there is an uncovered edge, there is a probability of at least $\frac{1}{m}$ to make an improvement, because adding the uncovered edge to $s$ reduces the number of uncovered edges to 0. This means that in expectation, it takes $m$ steps for RLS$_e$ to find this improvement.

When an edge, $e = \{v_1, v_2\}$, is deleted from the graph, if $e \notin E(s)$ then $s$ is still a maximal matching and corresponds to a 2-approximate solution. If $e \in E(s)$, then it is deleted from the solution as well. The new $s$ is still a matching but might be or not be a maximal matching. If $s$ is still a maximal matching then it is already a 2-approximation and we are done.

We examine the case where $s$ does not constitute a maximal matching anymore. If $s$ is not a maximal matching, then there is a non-empty set $E'$ such that:

$$E' = \{e_1 \mid e_1 \in E \wedge (\forall e_2 \in E(s) \Rightarrow e_1 \cap e_2 = \emptyset)\}.$$

Consider the set $E''$:

$$E'' = \{e_1 \mid e_1 \in E \wedge (\forall e_2 \in E(s), e_1 \cap e_2 = \emptyset) \wedge e_1 \cap e \neq \emptyset\}$$

The definition implies that $E'' \subseteq E'$. Here we show that $E' = E''$. If not, $\exists e'' \in E' \setminus E''$ which means that $e'' \cap e = \emptyset$ and $s$ was not covering $e''$ before removing $e$, and therefore was not a maximal matching which is in contrast to the given assumption on $s$.

Now we can define $U_1 = \{e_1 \mid e_1 \in E'' \wedge v_1 \in e_1\}$ and $U_2 = \{e_1 \mid e_1 \in E'' \wedge v_2 \in e_1\}$. We know that $U_1 \cap U_2 = \emptyset$ because the edge containing both $v_1$ and $v_2$ was $e$ which is deleted from the graph. Therefore, $U_1$ and $U_2$ define a partition over $E''$ and in order to cover edges in $E''$, edges from both of these sets need to be covered. All edges in $U_1$ include the node $v_1$ which implies that selecting any edge from $U_1$ covers all other edges from this set. Similarly selecting any edge from $U_2$ covers all edges from this set. Therefore, using RLS$_e$, at each step there is a probability of $\frac{|U_1|}{m}$ to cover all edges of $U_1$ and a probability of $\frac{|U_2|}{m}$ to cover all edges of $U_2$.

Note that any other move is rejected: No edge can be deleted from $s$, because $s$ is a matching and deleting any edge from it increases the number of uncovered edges; therefore is rejected (Lemma 8.23). Furthermore, all edges other than $e'' \in E''$ are covered by $s$ and adding them to $s$ results in a solution which is not a matching; hence, is rejected (Lemma 8.22).

With the mentioned probabilities of covering $U_1$ and $U_2$ in one step, each of them will be covered in expected time of $Cm$, where $C$ is a constant. Using Markov's inequality (Section 4.2.1) and $m^\epsilon$ phases of $Cm$, with probability $1 - e^{-\Omega(m^\epsilon)}$ all the uncovered edges will be covered in $O(m^{1+\epsilon})$. The new solution is a maximal matching which induces a 2-approximate solution. □

### 8.5.2 Analysis of (1+1) EA

In this section, we consider the (1+1) EA$_e$ and analyse maintaining a 2-approximate solution for the dynamic vertex cover problem for that. We have obtained new results for the dynamic change of adding an edge; but deleting an edge is more complicated to analyse. The reason is that the number of uncovered edges can be as large as $O(m)$ and when more than one flip can happen at each step, some uncovered edges can get covered but a smaller number of covered edges get uncovered. The best expected optimization time for this situation known so far is $O(m \log m)$ which is the same as the expected time of (1+1) EA$_e$ starting from scratch.

The following theorem, gives the result of our analysis for (1+1) $\text{EA}_e$ when edges are dynamically added to the graph.

**Theorem 8.26.** *Starting with a 2-approximate solution $s$, which is also a maximal matching for an instance of the problem, (1+1) $\text{EA}_e$ maintains the quality of the solution when one new edge is dynamically added to the graph in expected time of $O(m)$.*

*Proof.* The proof is quite similar to the proof of the first part of Theorem 8.25. When an edge is added to the graph, $s$ is still a matching, but it might be or not be a maximal matching. If $s$ is still a maximal matching then according to Lemmata 8.22 and 8.23 the algorithm can only replace $s$ by solutions that are also maximal matchings (solutions that are matchings and do not have uncovered edges); hence, a 2-approximate solution is induced from the solutions that the algorithm produces.

If $s$ is not a maximal matching, then only the new edge, $e$, might not be covered. Once again, according to Lemmata 8.22 and 8.23, the algorithm only accepts solutions that are matchings and have at most one uncovered edge. Moreover, while there is an uncovered edge, there is a probability of at least $\frac{1}{m}(1-\frac{1}{m})^{(m-1)}$ to make an improvement by adding that edge to $s$, which results in a complete cover and a 2-approximate solution. In other words, it takes expected time of at most $\frac{1}{\frac{1}{m}(1-\frac{1}{m})^{(m-1)}} = O(m)$ for (1+1) $\text{EA}_e$ to maintain 2-approximation when dynamic changes only include adding new edges. □

## 8.6 Conclusion

In this chapter, we have carried out rigorous runtime analyses on how the different evolutionary approaches already examined by Jansen et al. [70] for the static vertex cover problem, can deal with the dynamic version of the problem. We have investigated their three approaches together with (1+1) EA and simple randomised local search algorithms. For the first two examined approaches, we have presented classes of instances of bipartite graphs where adding edges lead to bad approximation behaviours even if the algorithms started with a 2-approximation. This shows that edge-based representation does not in general help with maintaining 2-approximations of the vertex cover problem, if the simple fitness function of the node-based approach is used. For the third approach in which the edge-based representation is used together with a fitness function that includes a large penalty for adjacent edges, we have shown that 2-approximations are maintained easily by recomputing maximal matchings of the dynamically changing graph.

# Chapter 9

# Obtaining 2-Approximations for the Weighted Vertex Cover Problem by Finding a Maximal Dual Solution

## 9.1 Introduction

In this chapter, we consider the weighted minimum vertex cover problem and investigate how its dual formulation can be exploited to design evolutionary algorithms that provably obtain a 2-approximation. Inspired by the work of Jansen et al. [70], we investigate a different way of approximating the minimum vertex cover problem by evolutionary algorithms. While Jansen et al. [70] considered the classical vertex cover problem, we analyse the weighted version of the problem. Although no direct connection to the use of dual formulations was made in [70], our investigations can be seen as a generalization of the approach based on matchings investigated in that paper. We study an edge-based encoding together with a multi-valued representation that works on the dual of the minimum vertex cover formulation. We are only aware of four previous theoretical works with multi-valued representations. Doerr et al. [33, 35] regard the optimization of multi-valued linear functions via a variant of the (1+1) EA. More recently, static and dynamically changing variants of multi-valued OneMax functions have been considered [31, 74].

Working with the dual formulation, our encoding assigns a weight to each edge. During the evolutionary process the weight of the edges may be increased or decreased and vertices whose constraints become tight are selected as vertices for the cover. We first study the situation where each weight can only increase or decrease by 1 at each

step and present pseudo-polynomial upper bounds on the expected time until our approaches have obtained a 2-approximation for the minimum vertex cover problem.

In order to deal with potentially large weights of the given graph, we incorporate *step size adaptation* into our algorithms. Step size adaptation is a popular mechanism to steer the progress of an evolutionary algorithm to the right level. Step size adaptation is a form of *parameter control* [40], where a parameter is changed during the execution of the algorithm. Adaptive parameters are very essential in continuous search spaces [12] and popularly used for covariance-matrix adaptation [54]. There are only few theoretical studies on adaptive parameters in discrete spaces. Dynamically choosing the number of parallel instances in parallel evolutionary algorithms is studied in [83], and self-adjusting of the number of bits to be flipped instead of a standard bit mutation is shown to improve the performance of the optimization process [32]. In other works it is shown that changing the mutation rate [13, 21, 29] can reduce the asymptotic runtime. In [29], the $(1 + (\lambda + \lambda))$ GA (proposed in [30]) is regarded, in which the mutation rate and the population size are correlated and the focus is on choosing the best population size during the process of optimization. Although their work is on discrete settings, Doerr and Doerr [29] have used the one-fifth success rule of step-size adaptation in evolutionary strategies. The one-fifth success rule says that if the probability of finding an improvement is greater than $1/5$ then the step-size should be increased; otherwise it should be decreased. Doerr and Doerr [29] have proved that when some conditions hold for the population size $\lambda$, the probability of finding an offspring of better fitness is larger than $1/5$. They reduce $\lambda$ by a constant factor $F > 1$ after finding an improvement, and increase it by a factor of $F^{1/4}$ after an iteration that did not improve the fitness. They have proved that with this settings, their genetic algorithm solves the One-Max problem in linear time.

In this chapter, defining $c_1 > 1$ and $c_2 > 1$ as two constants, we show that the use of step size adaptation where the step size is multiplied by $c_1$ in the case of a success and multiplied by $1/c_2$ in case of failure, leads to a polynomial upper bound on the expected runtime of the RLS algorithm to achieve a 2-approximation. Furthermore, we present a pseudo-polynomial lower bound for the (1+1) EA using this step size adaptation when $c_1 = c_2$. The proof uses the insight that the considered (1+1) EA is not able to achieve a sufficiently large step size during the optimization process in order to reach a 2-approximation. Note that this lower bound is proved under the condition that $c_1 = c_2$. The work of Doerr and Doerr [29] suggest that choosing these two constants in a way that the one-fifth rule holds may result in finding a polynomial upper bound on the optimization time of (1+1) EA. This problem is open for future work.

**Algorithm 22:** RLS

Initialize $s := 0^m$ and $\sigma := 1^m$;
**while** *termination condition not satisfied* **do**
  $s' := s$;
  Choose $i \in \{1, \cdots, m\}$ uniformly at random;
  Choose $b \in \{0, 1\}$ uniformly at random;
  **if** $b = 0$ **then**
    $s'_i := s'_i + \sigma_i$;
  **else**
    $s'_i := \max(s'_i - \sigma_i, 0)$;
  **if** $\sum_{i=1}^{m} s_i < \sum_{i=1}^{m} s'_i$ *and* $\sum_{j \in \{1,\cdots,m\} | e_j \cap \{v\} \neq \emptyset} s'_j \leq w(v),\ \forall v \in V$ **then**
    $s := s'$;
**return** $C := \{v \in V \mid w(v) = \sum_{j \in \{1,\cdots,m\} \mid e_j \cap \{v\} \neq \emptyset} s_j\}$;

This chapter is based on the paper that has been submitted to a conference [99] and is structured as follows. In Section 9.2, we present our edge-based approach based on a dual formulation for solving the minimum vertex cover problem. We analyze RLS and (1+1) EA with a step size of 1 in Section 9.3. Afterwards, we show a polynomial upper bound for RLS with Step Size Adaptation in Section 9.4 and a pseudo-polynomial lower bound for (1+1) EA with Step Size Adaptation in Section 9.5. Finally, we finish with some concluding remarks.

## 9.2 Preliminaries

In the weighted vertex cover problem, weights are assigned to vertices and the goal is to find a subset of nodes that covers all edges and has minimum weight. The formal definition of this problem is given in definition 3.4 of section 3.3. Here we consider the case where weights can only take integer values.

In this chapter, we analyse the behaviour of four evolutionary algorithms which find a 2-approximation for the weighted vertex cover problem using the concept of duality for this problem, that is explained in Section 3.3.2. It is worth to recall from Section 3.3.2 that 2-approximations of the VCP can be found by means of maximal solutions for the dual problem, which is a packing problem formulated as:

$$\begin{aligned} & \max \sum_{j=1}^{m} s_j \\ s.t. \quad & \sum_{j \in \{1,\cdots,m\} | e_j \cap \{v\} \neq \emptyset} s_j \leq w(v) \quad \forall v \in V \end{aligned}$$

**Algorithm 23:** (1+1) EA

Initialize $s := 0^m$ and $\sigma := 1^m$;
**while** *termination condition not satisfied* **do**
  $s' := s$;
  **for** $i := 1$ **to** $m$ **do**
    **with probability** $1/m$ **do**
      Choose $b \in \{0, 1\}$ uniformly at random;
      **if** $b = 0$ **then**
        $s'_i := s'_i + \sigma_i$;
      **else**
        $s'_i := \max(s'_i - \sigma_i, 0)$;
  **if** $\sum_{i=1}^{m} s_i < \sum_{i=1}^{m} s'_i$ *and* $\sum_{j \in \{1,\cdots,m\} | e_j \cap \{v\} \neq \emptyset} s'_j \leq w(v),\ \forall v \in V$ **then**
    $s := s'$;
**return** $C := \{v \in V \mid w(v) = \sum_{j \in \{1,\cdots,m\} \mid e_j \cap \{v\} \neq \emptyset} s_j\}$;

where $s_j \in \mathbb{N}^+$ denotes a weight on the edge $e_j$.

All the algorithms that we investigate in this chapter, find a 2-approximation vertex cover by means of a maximal dual solution that they find. A simple randomized local search (RLS) is presented in Algorithm 22, where a solution $s = (s_1, \cdots, s_m)$, is represented by a string of $m$ integers, denoting the weights of the $m$ edges of the input graph. This algorithm starts with the initial solution $s = 0^m$, and selects one edge at each step to increase or decrease the weight corresponding to that by one. The new solution replaces the old one, if the sum of weights of edges is increased, and the weight constraint of the packing problem is not violated for any of the vertices. At the end, the algorithm returns the set of nodes for which the constraint has become tight.

One other algorithm that we analyse in this paper is the (1+1) EA, presented in Algorithm 23, which is quite similar to the RLS of Algorithm 22 except for selecting the edges for mutation. In (1+1) EA, at each step a mutation happens on the weight of all edges with probability $1/m$ for each of them, while in RLS one edge is selected and the mutation takes place on the weight of that edge. Note that in (1+1) EA more than one mutation may happen on the current solution.

In both RLS and (1+1) EA (Algorithms 22 and 23) the increment size of one on the weights of the edges might be too small and make the algorithm slow. Motivated by step size adaptation in evolution strategies [12] in RLS with Step Size Adaptation and (1+1) EA with Step Size Adaptation (Algorithms 24 and 25), a step size for each edge is kept in an auxiliary vector $\sigma = (\sigma_1, \cdots, \sigma_m)$. The initial step size for all edges is set to 1. The algorithms work with two constant parameters $c_1 > 1$ and $c_2 > 1$. If a mutation

**Algorithm 24:** RLS with Step Size Adaptation

Initialize $s := 0^m$ and $\sigma := 1^m$;
**while** *termination condition not satisfied* **do**
  $s' := s$;
  $I := \emptyset$;
  Choose $i \in \{1, \cdots, m\}$ uniformly at random;
  Choose $b \in \{0, 1\}$ uniformly at random;
  **if** $b = 0$ **then**
    $s'_i := s'_i + \sigma_i$;
  **else**
    $s'_i := \max(s'_i - \sigma_i, 0)$;
  $I := I \cup \{i\}$;
  **if** $\sum_{i=1}^{m} s_i < \sum_{i=1}^{m} s'_i$ *and* $\sum_{j \in \{1,\cdots,m\} | e_j \cap \{v\} \neq \emptyset} s'_j \leq w(v),\ \forall v \in V$ **then**
    $s := s'$;
    $\sigma_i := c_1 \cdot \sigma_i,\ \forall i \in I$;
  **else**
    $\sigma_i := \max\left(\frac{\sigma_i}{c_2}, 1\right),\ \forall i \in I$;
**return** $C := \{v \in V \mid w(v) = \sum_{j \in \{1,\cdots,m\} \mid e_j \cap \{v\} \neq \emptyset} s_j\}$;

with that size is accepted, the step size is increased by a factor of $c_1$; otherwise, it is decreased by a factor of $c_2$ with a minimum accepted size of one.

Analysing the runtime of our algorithms, we find the number of iterations of the while loop, until a maximal packing solution is found, which induces a complete vertex cover. We call this the expected time of obtaining the desired goal by the considered algorithm. It should be noted that the edge-based approach for the unweighted minimum vertex cover investigated by Jansen et al. [70] can be seen as a special case of our formulation as the use of maximal matchings is equivalent to the dual problem if all edges have a weight of 1.

## 9.3 RLS and (1+1) EA

In this section, we present the analysis on finding 2-approximations for the weighted vertex cover problem by RLS and (1+1) EA.

**Theorem 9.1.** *The expected time of RLS and (1+1) EA (Algorithms 22 and 23) to find a 2-approximation is* $O(m \cdot OPT)$.

*Proof.* In order to prove this theorem, we show that the algorithms find a maximal solution for the dual problem in expected time $O(m \cdot OPT)$. Having achieved that

**Algorithm 25:** (1+1) EA with Step Size Adaptation

Initialize $s := 0^m$ and $\sigma := 1^m$;
**while** *termination condition not satisfied* **do**
  $s' := s$;
  $I := \emptyset$;
  **for** $i := 1$ **to** $m$ **do**
    **with probability** $1/m$ **do**
      Choose $b \in \{0, 1\}$ uniformly at random;
      **if** $b = 0$ **then**
        $s'_i := s'_i + \sigma_i$;
      **else**
        $s'_i := \max(s'_i - \sigma_i, 0)$;
      $I := I \cup \{i\}$;
  **if** $\sum_{i=1}^{m} s_i < \sum_{i=1}^{m} s'_i$ *and* $\sum_{j \in \{1,\cdots,m\} | e_j \cap \{v\} \neq \emptyset} s'_j \leq w(v),\ \forall v \in V$ **then**
    $s := s'$;
    $\sigma_i := c_1 \cdot \sigma_i,\ \forall i \in I$;
  **else**
    $\sigma_i := \max\left(\frac{\sigma_i}{c_2}, 1\right),\ \forall i \in I$;
**return** $C := \{v \in V \mid w(v) = \sum_{j \in \{1,\cdots,m\} \mid e_j \cap \{v\} \neq \emptyset} s_j\}$;

maximal solution, the algorithms return the set

$$C := \{v \in V \mid w(v) = \sum_{j \in \{1,\cdots,m\} \mid e_j \cap \{v\} \neq \emptyset} s_j\}$$

as the solution for the weighted vertex cover problem which, according to Theorem 3.6, is a 2-approximation of the optimal solution.

If a solution $s$ is not a maximal solution for the dual problem, then there exists at least one edge for which the assigned weight can be increased. The probability of selecting only that edge for mutation and choosing $b = 0$ is at least $\frac{1}{2 \cdot m}$ for RLS and $\frac{1}{2 \cdot e \cdot m}$ for (1+1) EA at each step, and according to the Weak Duality Theorem (Theorem 3.5), the cost of any maximal solution is upper bounded by $OPT$. Therefore, using the method of Fitness Based Partitions [116], we find the expected time $O(m \cdot OPT)$ for finding a maximal solution for the dual problem by both algorithms. □

Note that the presented upper bound in Theorem 9.1 is a pseudo polynomial time, because $OPT$ can be exponentially large with respect to the input size. In the remainder of this section, we introduce an instance of the problem for which a pseudo polynomial time is required for finding a 2-approximation. This instance is also used in Section 9.5, as a hard instance for the (1+1) EA with Step Size Adaptation.

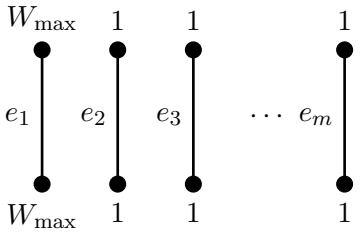


FIGURE 9.1: $G$, a hard instance for RLS and (1+1) EA

The hard instance of the problem, $G$, illustrated in Figure 9.1, contains $m$ edges, $e_1, \cdots, e_m$, none of which share a node with another. One of the edges, $e_1$, is adjacent to two nodes of weight $W_{\max}$ while all other edges are adjacent to vertices of weight 1. The dual problem of this instance has only one maximal solution: $s_1 = W_{\max}$ and $s_i = 1,\ 2 \leq i \leq m$. In this instance, we assume that $W_{\max} > 2^m$.

**Theorem 9.2.** *With probability $1 - e^{-\Omega(2^m)}$, the required time for RLS and the (1+1) EA (Algorithm 22 and 23) to find a 2-approximation of $G$ is lower bounded by $\Omega(m \cdot W_{\max})$.*

*Proof.* Consider a phase of $\frac{m \cdot W_{\max}}{4}$ steps. Let $X$ be the number of times that $e_1$ is selected for mutation by RLS or (1+1) EA in this phase. Since the probability of selecting $e_1$ is $\frac{1}{m}$ for both algorithms, the expected value of $X$ is $\frac{W_{\max}}{4}$. As these probabilities are independent of each other at each step, by Chernoff bounds we get

$$\Pr(X > \frac{W_{\max}}{2}) \leq e^{-\frac{W_{\max}}{12}} = e^{-\Omega(2^m)}$$

At each step that $e_1$ is selected for mutation, $s_1$ can be increased by at most 1. Therefore, with probability $1 - e^{-\Omega(2^m)}$, in a phase of $\frac{m \cdot W_{\max}}{4} = \Omega(m \cdot W_{\max})$ steps, we have $s_1 \leq \frac{W_{\max}}{2}$, i.e. $s_1$ does not reach its maximal value of $W_{\max}$. Therefore, with probability $1 - e^{-\Omega(2^m)}$, the RLS and the (1+1) EA find a 2-approximation of $G$ in time $\Omega(m \cdot W_{\max})$. □

## 9.4 RLS with Step Size Adaptation

In this section, we analyse the behaviour of RLS with Step Size Adaptation for finding 2-approximations of the weighted vertex cover problem. We prove that the RLS with Step Size Adaptation finds a 2-approximation for the weighted vertex cover problem in expected polynomial time with respect to the input size, provided that $c_1 = c_2$. This also holds for $c_1 \geq c_2$, which is stated in Corollary 9.6. The two lemmata below are used in the proof of the main result stated later.

**Lemma 9.3.** *If $c_1 = c_2$, the step size $\sigma_i$ for each edge $e_i$ in RLS with Step Size Adaptation, can only take a value from*

$$\{{c_1}^k \mid 0 \leq k \leq \lceil \log_{c_1} W_{\max} \rceil\},$$

*where $W_{\max}$ is the largest weight assigned to any vertex.*

*Proof.* The algorithm starts with initial value of $\sigma_i = 1$ for all edges. This value is increased by a factor of $c_1$ each time a mutation is accepted for edge $e_i$, and is divided by the same factor with a minimum accepted value of one if the mutation is rejected (lines 15 and 17 of Algorithm 24). Therefore $\sigma_i$ is always a power of $c_1$. Moreover, in order to fulfil the constraints on the vertices, none of the edges can be assigned a weight larger than $W_{\max}$. Therefore, any mutation that increases the current weight of an edge by at least $W_{\max}$, is rejected. Therefore, $\sigma_i$ can be increased to at most $c_1^k$ where $k = \lceil \log_{c_1} W_{\max} \rceil$. □

**Lemma 9.4.** *For an edge $e_i$, let $D(s_i) = MAX_i - s_i$ where $s$ is the solution obtained so far by the algorithm and $MAX_i$ is the maximum acceptable value for $s_i$ in the current solution $s$. In expected time $O(m \log_{c_1}^2 W_{\max})$ a solution $s'$ with $D(s'_i) \leq \frac{c_1 \cdot D(s_i)}{c_1+1}$ is found by RLS with Step Size Adaptation when $c_1 = c_2$.*

*Proof.* Note that since at any step only one mutation happens, for any solution $s'$ obtained after $s$, we have $D(s'_i) \leq D(s_i)$, otherwise the algorithm would have rejected $s'$. We divide the analysis into two phases. The first phase, consists of all steps until the algorithm reaches a situation in which $s_i$ is selected for an increasing mutation and $\sigma_i \leq D(s_i)$. In this phase $\sigma_i$ decreases. The second phase begins when $\sigma_i$ starts increasing. We show that by the end of the second phase, we have reached a solution $s'$ with $D(s'_i) \leq c_1 \cdot D(s_i)/(c_1 + 1)$.

In the first phase, whenever $s_i$ is selected for an increase, we have $\sigma_i > D(s_i)$; therefore, $\sigma_i$ is decreased. If $\sigma_i \leq D(s_i)$ at a step in which $s_i$ is selected for an increase, then we are already in the second phase and $\sigma_i$ is added to $s_i$, resulting in decreasing $D(s_i)$. Note that it is not only increasing $s_i$ that decreases $D(s_i)$. Instead, increasing the weight of other edges that are adjacent to $e_i$ can also decrease $D(s_i)$. If we reach a solution $s'$ where $D(s'_i) = 0$ in Phase 1, then we already have $D(s'_i) \leq \frac{c_1 \cdot D(s_i)}{c_1+1}$ (stated in the lemma) without going to Phase 2.

Here we show that Phase 1 is over in expected time $O(m \cdot \log_{c_1} W_{\max})$. At each step, with probability $\frac{1}{m}$, $s_i$ is mutated. Since $\sigma_i > D(s_i)$, increasing mutations on $s_i$ are rejected as well as decreasing mutations, and $\sigma_i$ is divided by $c_1$ with each rejection. This needs to be done at most $\log_{c_1} W_{\max}$ times until we reach $\sigma_i \leq D(s_i)$, which in expectation takes $O(m \cdot \log_{c_1} W_{\max})$.

The second phase starts when we reach a step with an increasing mutation on $s_i$ in which $1 \leq \sigma_i \leq D(s_i)$. This move is accepted and $\sigma_i$ is increased by a factor of $c_1$. Note that $D(s_i)$ might be far larger than $\sigma_i$. Since $\sigma_i$ is always a power of $c_1$, we define

$a \in \mathbb{N}^+$ as $a = \log_{c_1} \sigma_i$ to make the proof easier. Due to Lemma 9.3, we have $0 \leq a \leq \lceil \log_{c_1} W_{\max} \rceil$. Here, an increase on $s_i$ is accepted by the algorithm and $a$ is increased to $a+1$, while a decrease is rejected and $a$ is decreased to $a-1$. The increase and decrease happen with equal probability; therefore, a fair random walk happens for $a$ on integer values in $[0, \lceil \log_{c_1} W_{\max} \rceil]$, with initial value of at least 0.

It is proved that the expected number of required steps for a fair random walk to visit all vertices in a graph with $v$ vertices and $e$ edges is bounded by $2e(v-1)$ [2]. In the fair random walk that happens on $a$, there are $\lceil \log_{c_1} W_{\max} \rceil + 1$ vertices to visit with $\lceil \log_{c_1} W_{\max} \rceil$ edges between them. This gives us the expected number of steps $k = 2\lceil \log_{c_1} W_{\max} \rceil^2 = O(\log^2_{c_1} W_{\max})$ for our random walk, to reach any possible value of $a$. As a result, as long as $\sigma_i \leq D(s_i)$ holds, in $k$ mutations on $s_i$, $a$ reaches its maximal possible value which is upper bounded by $\lceil \log_{c_1} W_{\max} \rceil$ after which the inequality does not hold. This implies that $k$ is an upper bound on the number of mutations that can happen on $s_i$ before this phase ends, which is in expectation done in time $O(m \cdot \log^2_{c_1} W_{\max})$. At the end of this phase, $\sigma_i > D(s'_i)$, whereas the last accepted mutation has increased $s'_i$ by at least $\frac{1}{c_1}\sigma_i$. This implies that

$$D(s'_i) \leq D(s_i) - \frac{1}{c_1}\sigma_i \leq \frac{c_1}{c_1+1} D(s_i),$$

which completes the proof. □

**Theorem 9.5.** *The RLS with Step Size Adaptation with $c_1 = c_2$ and the initial solution $s = 0^m$, finds a vertex cover that is at least a 2-approximation in expected time $O(m \cdot \log^3_{c_1} W_{\max})$.*

*Proof.* Similar to the proof of Theorem 9.1, we show that the algorithm finds a maximal solution for the dual problem in expected time $O(m \cdot \log^3_{c_1} W_{\max})$.

For each edge $e_i$, the distance of $s_i$ to its maximal value, $D_i$, is decreased by at least $\frac{D_i}{c_1+1}$ by RLS with Step Size Adaptation, in expected time $O(m \log^2_{c_1} W_{\max})$ according to Lemma 9.4. Since the initial value of $D_i$ is upper bounded by $W_{\max}$, according to Multiplicative Drift Theorem [34], $s_i$ reaches its maximal value in expected time $O(m \log^3_{c_1} W_{\max})$. □

In the proof of Lemma 9.4, setting $c_1 > c_2$, is in favour of increasing the value of $\sigma_i$; therefore, the lemma holds in that situation as well, resulting in the following corollary.

**Corollary 9.6.** *The RLS with Step Size Adaptation with $c_1 \geq c_2$ and the initial solution $s = 0^m$, finds a vertex cover that is a 2-approximation in expected time $O(m \cdot \log^3_{c_1} W_{\max})$.*

## 9.5 (1+1) EA with Step Size Adaptation

In this section we prove a pseudo polynomial lower bound on the time that (1+1) EA with Step Size Adaptation requires for finding a 2-approximation of the weighted vertex cover problem, when $c_1 \leq c_2$. To prove this lower bound, we investigate the behaviour of (1+1) EA with Step Size Adaptation on $G$ (Figure 9.1), the hard instance of the problem presented in Section 9.3, with the assumption that $W_{\max} \geq {c_1}^m$. We show that with high probability, the (1+1) EA with Step Size Adaptation needs exponential time with respect to the input size for finding a maximal dual solution for $G$.

In the following, $A(s) = \{s_i \mid s_i = 1,\ 2 \leq i \leq m\}$. Moreover, Phase 1 indicates the steps starting from the initial step until finding a solution $s$, with $|A(s)| \geq \frac{3m}{4}$, and Phase 2 consists of ${c_1}^{m^{\epsilon/2}}$ steps, where $0 < \epsilon \leq \frac{1}{3}$, starting by the end of Phase 1. We also define Property 9.7 below, which is used in Lemmata 9.9 and 9.11, and Theorem 9.12.

*Property* 9.7. For current solution $s$, we have $|A(s)| \geq \frac{m}{2}$.

In order to prove the main theorem of this section, we make use of Lemmata 9.8, 9.9, 9.10 and 9.11, which follow.

**Lemma 9.8.** *For sufficiently large $m$, with probability $1-e^{-\Omega(m^\epsilon)}$, Phase 1 needs at most $m^{1+\epsilon}$ steps, where $\epsilon > 0$ is a constant.*

*Proof.* Let $Z(s) = \{s_i \mid s_i = 0,\ 2 \leq i \leq m\}$. Note that $|Z(s)| + |A(s)| = m - 1$. At each step, if one of the edges of set $Z(s)$ is selected for a mutation of increase, and no other mutations happen, the new solution is accepted by the algorithm. Therefore, the probability of producing a solution $s'$ with $|A(s')| = |A(s)| + 1$ is at least

$$\frac{|Z(s)|}{2 \cdot e \cdot m} = \frac{m - 1 - |A(s)|}{2 \cdot e \cdot m}.$$

This implies that, the positive drift on $|A(s)|$, denoted by $\Delta_+$, is at least $\frac{m-1-|A(s)|}{2 \cdot e \cdot m}$ at each step.

Moreover, to obtain a solution $s'$ with $|A(s')| = |A(s)| - k$ from $s$, $k$ mutations should happen on edges of $A$, and in order to make these changes acceptable, a mutation of increase should happen on $s_1$. The probability of increasing $s_1$ at each step is $\frac{1}{2m}$, and the probability of $k$ other mutations to happen at the same step is upper bounded by

$$\binom{m-1}{k} \cdot \left(\frac{1}{m}\right)^k \left(1 - \frac{1}{m}\right)^{m-1-k} \leq \frac{1.06}{k!e},$$

for sufficiently large $m$. Here, it suffices if we assume $m \geq 20$. Overall, the probability of finding a solution $s'$ with $|A(s')| = |A(s)| - k$ is at most $\frac{1.06}{k!e \cdot 2m}$. As a result, for the negative drift on $|A(s)|$, denoted by $\Delta_-$, we have

$$\begin{aligned}\Delta_- &\leq \sum_{k=1}^{|A(s)|} k \cdot \frac{1.06}{k!e \cdot 2m} \\ &= \frac{1.06}{e \cdot 2m} \sum_{k=1}^{|A(s)|} \frac{1}{(k-1)!} \\ &\leq \frac{1.06}{e \cdot 2m} \cdot 3 = \frac{3.18}{e \cdot 2m}.\end{aligned}$$

Summing up, the total drift on $|A(s)|$ is

$$\Delta = \Delta_+ - \Delta_- \geq \frac{m - 4.18 - |A(s)|}{2 \cdot e \cdot m}.$$

We now analyse the time to find a solution with $|A(s)| \geq \frac{3m}{4}$. For any solution $s$ with $|A(s)| < \frac{3m}{4}$, we have $\Delta \geq \frac{\frac{m}{4}-4}{2 \cdot e \cdot m} \geq 0.0075$, since we have assumed $m > 20$. By additive drift argument [57], we can see that a solution with $|A(s)| \geq \frac{3m}{4}$ is found in expected time $\frac{1}{0.0075} \cdot \frac{3m}{4} = 100m$. By Markov's inequality, with probability at least $\frac{1}{2}$, the time until finding that solution is at most $200m$. Therefore, in a phase of $m^{1+\epsilon}$ steps, the probability of not finding that solution is $(\frac{1}{2})^{\frac{m^\epsilon}{200}} = e^{-\Omega(m^\epsilon)}$. □

**Lemma 9.9.** *For sufficiently large $m$, with probability $1 - e^{-\Omega(m)}$, Property 9.7 holds during Phase 2.*

*Proof.* For proving this lemma, we use the Simplified Drift Theorem (Theorem 4.7) presented in Section 4.5.3. We analyse the changes on the size of $A(s)$, and no filtration is applied on the steps.

Phase 2 starts with the solution $s$ with $|A(s)| \geq \frac{3m}{4}$, found by the end of Phase 1. Using Simplified Drift Theorem with parameters $\delta = 1$, $r(l) = 1$ and interval $[a, b] = [\frac{m}{2}, \frac{3m}{4}]$, we show that with high probability, a solution $s'$ with $|A(s')| \leq \frac{m}{2}$ is not found by the algorithm until end of Phase 2.

Let $X_t = |A(s)|$, where $s$ is the solution obtained at time $t$. The total drift on the value of $X_t$ is $\Delta$ of the proof of Lemma 9.8, which is at least $0.0075$ when $X_t \leq \frac{3m}{4}$. Therefore, the two conditions of the Simplified Drift Theorem hold:

1. $\mathrm{E}(X_{t+1} - X_t \mid a \leq X_t \leq b) = \mathrm{E}(X_{t+1} - X_t \mid \frac{m}{2} \leq X_t \leq \frac{3m}{4}) \geq 0.0075$, and

2. $\Pr(|X_{t+1} - X_t| \geq j \mid a \leq X_t) \leq \frac{1}{j!e} \leq \frac{1}{2^j} = \frac{r(l)}{(1+\delta)^j}$

The inequality regarding the second condition holds, because the probability of mutating $j$ edges at one step follows the Poisson distribution and is $\frac{1}{j!e}$. Having these two conditions satisfied, the Simplified Drift Theorem says that the probability of finding a solution with $|A(s)| \leq \frac{m}{2}$ in time $2^{\frac{c^* m}{4}}$, $c^* > 0$ a constant, is at most $2^{-\Omega(\frac{m}{4})}$. This implies that with probability $1 - e^{-\Omega(m)}$, such a solution is not found by the end of Phase 2 which consists of ${c_1}^{m^{\epsilon/2}} = 2^{\log_2 c_1 \cdot m^{\epsilon/2}}$ steps. □

**Lemma 9.10.** *Let $\epsilon \leq 1/3$ be a positive constant. In Phase 1, with probability $1 - e^{-\Omega(m^\epsilon)}$, the (1+1) EA with Step Size Adaptation does not reach a solution where $s_1 > 2 \cdot m^\epsilon \cdot {c_1}^{2 \cdot m^\epsilon}$. Moreover, the step size of $s_1$ does not exceed ${c_1}^{2 \cdot m^\epsilon}$, i.e. $\sigma_1 \leq {c_1}^{2 \cdot m^\epsilon}$.*

*Proof.* From Lemma 9.8, we know that this phase is at most $m^{1+\epsilon}$ steps. Let $X$ be the number of times that the first edge is selected for mutation during Phase 1. Since the probability of selecting each edge at each step is $\frac{1}{m}$, the expected value of $X$ is at most $m^\epsilon$. Moreover, since probability of selecting edges are independent of each other, by Chernoff bounds we have:

$$\Pr(X \geq 2 \cdot m^\epsilon) \leq e^{-m^\epsilon/3}.$$

Therefore, with probability $1 - e^{-\Omega(m^\epsilon)}$ the first edge is not selected for mutation more than $2 \cdot m^\epsilon$ times, which means that the step size for that edge is at most ${c_1}^{2 \cdot m^\epsilon}$ after that phase. This implies that $2 \cdot m^\epsilon \cdot {c_1}^{2 \cdot m^\epsilon}$ is an upper bound for the value of $s_1$ by the end of Phase 1. Note that $s_1$ and $\sigma_1$ have not reached their maximal values, since $\epsilon \leq 1/3$. □

In the following lemma, we show that when $|A(S)| \geq m/2$, the probability of decreasing the step size $\sigma_1$ is larger than the probability of increasing it. This lemma is used in Theorem 9.12 to show that we do not reach large values of $\sigma_1$ in polynomial time.

**Lemma 9.11.** *Assuming that Property 9.7 holds, and also assuming that $\sigma_1 > m$ and $s_1 \leq W_{\max}$, at any step where $\sigma_1$ is changed by (1+1) EA with Step Size Adaptation, it is increased with probability $P_{inc} < 0.4$ and decreased with probability $P_{dec} > 0.6$.*

*Proof.* The value of $\sigma_1$ changes at the steps where $e_1$ is selected for mutation. All other steps make no change on $\sigma_1$. Here we only consider the steps at which $e_1$ is selected for a mutation.

The value of $\sigma_1$ increases when a mutation on $e_1$ is accepted. Since $\sigma_1 > m$ and $s_1 \leq W_{\max}$, any mutation that decreases the value of $e_1$ is rejected. Since we have assumed

that Property 9.7 holds, there are at least $\frac{m}{2}$ edges other than $e_1$, with a weight of one. A mutation of increase on these edges is rejected. Therefore, an increase on $e_1$ is also rejected if one of those edges is selected for an increase in addition to $e_i$ at the same step. The probability that an increase is selected to be done on $e_1$, while none of those edges are selected for increase, is:

$$\frac{1}{2} \cdot \left(1 - \frac{1}{2m}\right)^{m-\frac{m}{2}} \leq \frac{1}{2} \cdot \left(\frac{1}{e}\right)^{\frac{1}{4}} < 0.4$$

This probability is an upper bound for the probability that an acceptable increase on $e_1$ happens, which is denoted by $P_{inc}$. In other words:

$$P_{inc} < 0.4$$

Since $P_{inc} + P_{dec} = 1$ at steps where a mutation happens on $e_1$, we have $P_{dec} > 0.6$. □

**Theorem 9.12.** *For sufficiently large $m$ and a positive constant $\epsilon \leq \frac{1}{3}$, with probability $1 - e^{-\Omega(m^{\epsilon/2})}$, the required time for (1+1) EA with Step Size Adaptation (Algorithm 25) to find a 2-approximation on $G$ with $W_{\max} = {c_1}^m$ is lower bounded by $2^{m^{\epsilon/2}}$, when $c_1 = c_2$.*

*Proof.* According to Lemma 9.10, during Phase 1, with probability $1 - e^{-\Omega(m^{\epsilon})}$, we have $\sigma_1 \leq {c_1}^{2 \cdot m^{\epsilon}}$. Using Lemma 9.11 and the Gambler's Ruin Theorem (Theorem 4.3 of Section 4.4.2), we prove that with high probability, in Phase 2, we always have $\sigma_1 \leq {c_1}^{m^{2\epsilon}}$.

Due to Lemma 9.9, with probability $1 - e^{-\Omega(m)}$, Property 9.7 holds during Phase 2 which is a requirement of Lemma 9.11. However, Lemma 9.11 can only be used for the steps where $\sigma_1 > m$, while Phase 2 may start with $\sigma_1 \leq m$. Nevertheless, in order to reach large values of ${c_1}^{m^{2\epsilon}}$ or greater, at some point of Phase 2, we need to deal with a situation where $m < \sigma_1 \leq c_1 m$, since $\sigma_1$ increases at each step at most by a factor of $c_1$. According to Lemma 9.11, at the steps in which $e_1$ is selected for mutation, the probability of increasing $\sigma_1$ is $p \leq 0.4$ and the probability of decreasing it is $q \geq 0.6$.

Let $\sigma_1^0$ be the value of $\sigma_1$ at the first point in Phase 2 where $m < \sigma_1 \leq c_1 m$. If $\sigma_1 \leq c_1 m$ holds, then for sufficiently large $m$ we also have $\sigma_1 \leq {c_1}^{2 \cdot m^{\epsilon}}$. Starting from that point where $m < \sigma_1 \leq {c_1}^{2 \cdot m^{\epsilon}}$, we investigate whether the algorithm reaches a situation where $\sigma_1 \leq m$ earlier than a situation where $\sigma_1 \geq {c_1}^{m^{2\epsilon}}$.

Every time $\sigma_1$ is increased, it is increased by a factor of $c_1$ and every time that it is decreased, it is decreased by a factor of $c_2$. Since we have assumed that $c_1 = c_2$, one increasing step and one decreasing step cancel each other and the problem can

be mapped to the problem of Gambler's Ruin Theorem (Theorem 4.3) with parameters $p$ and $q$ described above and $\delta = \frac{q}{p} \geq \frac{0.6}{0.4} > 1$. The number of times that $\sigma_1 = \sigma_1^0$ needs to be decreased to reach $\sigma_1 \leq m$ is at most

$$\lceil \log_{c_2}(\sigma_1^0/m) \rceil \leq \log_{c_2} \left( \frac{{c_1}^{2 \cdot m^\epsilon}}{m} \right) + 1 \leq 2 \cdot m^\epsilon + 1$$

Also, the number of times that $\sigma_1 \leq m$ needs to be increased to reach $\sigma_1 \geq {c_1}^{m^{2\epsilon}}$ is at least

$$\lceil \log_{c_1}({c_1}^{m^{2\epsilon}}/m) \rceil \geq m^{2\epsilon} - \lfloor \log_{c_1} m \rfloor$$

Therefore, other parameters of the Gambler's Ruin Theorem would be $x \leq 2 \cdot m^\epsilon + 1$ and $z \geq m^{2\epsilon} - \lfloor \log_{c_1} m \rfloor$. Using that theorem, we get $P_x$, the probability of reaching a state where $\sigma_1 \geq {c_1}^{m^{2\epsilon}}$ before reaching a state where $\sigma_1 \leq m$ as:

$$P_x = \frac{(\delta)^x - 1}{(\delta)^z - 1} \leq \frac{(\delta)^{2 \cdot m^\epsilon + 1} - 1}{(\delta)^{m^{2\epsilon} - \lfloor \log_{c_1} m \rfloor} - 1} = e^{-\Omega(m^\epsilon)}.$$

Consider a phase of $2^{m^{\epsilon/2}}$ steps. We here show that with probability $e^{-\Omega(m^{\epsilon/2})}$, $\sigma_1 \geq {c_1}^{m^{2\epsilon}}$ during this phase.

We saw that with probability $1 - e^{-\Omega(m^\epsilon)}$ we reach a state where $\sigma_1 \leq m$ before a state where $\sigma_1 \geq {c_1}^{m^{2\epsilon}}$. If $\sigma_1$ never increases to $c_1^{m^\epsilon}$ after that, then we never reach a state where $\sigma_1 \geq {c_1}^{m^{2\epsilon}}$. Otherwise, it spends at least

$$\lceil \log_{c_1}({c_1}^{m^\epsilon}/m) \rceil = m^\epsilon - \lfloor \log_{c_1} m \rfloor$$

steps to reach $c_1^{m^\epsilon}$. In a phase of $2^{m^{\epsilon/2}}$ steps, there are at most

$$k = \frac{2^{m^{\epsilon/2}}}{m^\epsilon - \lfloor \log_{c_1} m \rfloor}$$

times that $\sigma_1$ increases to $c_1^{m^\epsilon}$, and probability of reaching ${c_1}^{m^{2\epsilon}}$ from there is only $e^{-\Omega(m^\epsilon)}$. Therefore, the probability of $\sigma_1$ to reach ${c_1}^{m^{2\epsilon}}$ at least once in a phase of $2^{m^{\epsilon/2}}$ steps, is at most

$$k \cdot e^{-\Omega(m^\epsilon)} = e^{-\Omega(m^{\epsilon/2})}.$$

So far we have proved that with probability $1 - e^{-\Omega(m^{\epsilon/2})}$, $\sigma_1 \leq {c_1}^{m^{2\epsilon}}$ during Phase 2 which consists of ${c_1}^{m^{\epsilon/2}}$ steps. Moreover, according to Lemma 9.10, with probability $1 - e^{-\Omega(m^\epsilon)}$, we have $s_1 \leq 2 \cdot m^\epsilon \cdot {c_1}^{2m^\epsilon}$ by the end of Phase 1. Therefore, the value of $s_1$

during both phases is always upper bounded by

$$2 \cdot m^{\epsilon} \cdot {c_1}^{2m^{\epsilon}} + {c_1}^{m^{\epsilon/2}} \cdot {c_1}^{m^{2\epsilon}}$$

which is less than $W_{\max}$, since $\epsilon \leq 1/3$. Therefore, with probability $1 - e^{-\Omega(m^{\epsilon/2})}$, the (1+1) EA with Step Size Adaptation does not find a 2-approximation in time $2^{m^{\epsilon/2}}$. □

Note that for $c_1 < c_2$, the probability of reaching a situation where $\sigma_1 \geq {c_1}^{m^{2\epsilon}}$ before reaching $\sigma_1 < m$ is even smaller, since the number of increasing steps that are required to cancel one decreasing step is greater than one. Therefore, this situation is in favour of reaching $\sigma_1 \leq m$, resulting in the following corollary.

**Corollary 9.13.** *For sufficiently large $m$ and a positive constant $\epsilon \leq \frac{1}{3}$, with probability $1 - e^{-\Omega(m^{\epsilon/2})}$, the required time for (1+1) EA with Step Size Adaptation (Algorithm 25) to find a 2-approximation of $G$ is lower bounded by $2^{m^{\epsilon/2}}$, when $c_1 \leq c_2$.*

## 9.6 Conclusion

In this chapter, we have considered how to solve the minimum vertex cover problem by its dual formulation based on a multi-valued edge-based encoding. We have proven pseudo-polynomial upper bounds for RLS and the (1+1) EA until they have achieved a 2-approximation. Furthermore, we have investigated the use of step-size adaptation in both algorithms and shown that RLS with step size adaptation obtains a 2-approximation in expected polynomial time; whereas the corresponding (1+1) EA still encounters a pseudo-polynomial lower bound.

# Chapter 10

# Conclusion

In this thesis, we have performed theoretical analyses of using local search and evolutionary algorithms for two combinatorial optimization problems, namely the generalized travelling salesman problem and the vertex cover problem.

We started by introducing local search and evolutionary algorithms in Chapter 2, together with describing multi-objectiveness and $\varepsilon-$dominance, a diversity mechanism that helps the algorithm keep the population size polynomial. In Chapter 3 we presented the formal definition of the generalised travelling salesman problem and the vertex cover problem, as well as an introduction to the concept of duality and hierarchical optimization. We have continued with Chapter 4 in which we have presented some of the state-of-the-art techniques that are widely used in the field of runtime analysis for combinatorial optimization problems. We have also presented a brief description on parameterized complexity analysis and FPT algorithms.

In Chapter 5, we have investigated local search methods and provided parameterized complexity analysis for simple evolutionary algorithms based on two hierarchical approaches for the generalized travelling salesman problem. We have gained new insights into the complimentary abilities of local search algorithms based on the two hierarchical approaches. We have proven that there are instances that can be solved in polynomial time with one approach, while the other approach fails to find an optimal solution. Furthermore, we have presented and analysed a class of instances where combining the two approaches into a variable-neighbourhood search helps to escape from local optima of the single approaches. After investigating local search algorithms, we have proven lower and upper bounds for optimization time of (1+1)EA with both approaches in Chapter 6. Our analyses show that the (1+1)EA with the first approach is a fixed parameter tractable evolutionary algorithm with respect to the number of clusters, while the same does not hold for the second approach. However, we have proven

that there are instances that (1+1)EA with the second approach solves to optimality in polynomial time, while the first approach needs exponential time.

After the analyses of generalized travelling salesman problem, we have investigated the behaviour of local search and evolutionary algorithms on the vertex cover problem. It had already been shown that a simple multi-objective evolutionary algorithm with a problem specific mutation operator is a fixed parameter evolutionary algorithm for the classical version of this problem and finds 2-approximations in expected polynomial time [77]. We have extended this analysis to the weighted version of the vertex cover problem in Chapter 7. We have proven upper bounds for the expected optimization time of Global SEMO; showing that this algorithm efficiently computes a 2-approximation as long as the value of an optimal solution is small. Furthermore, we have studied the algorithm DEMO, a simple multi-objective evolutionary algorithm that uses the $\varepsilon$-dominance technique as a diversity mechanism, and proven that the population size is polynomially bounded with this technique. Consequently, we have proven that this algorithm reaches a 2-approximation in expected polynomial time.

Theoretical analysis of simple evolutionary algorithms on the vertex cover problem has continued in this thesis with Chapter 8, in which the dynamic version of the problem is investigated. It had already been shown in the literature that a 2-approximation can be obtained by using an edge-based representation in a simple evolutionary algorithm combined with a specific fitness function that includes a large penalty for adjacent edges [70]. We have investigated the node-based representation with a simple fitness function, and the edge-based representation with simple and specific fitness functions according to their approximation behaviour in the dynamic setting. For the node-based and edge-based approach with standard fitness function, we have presented classes of instances of bipartite graphs where adding edges lead to bad approximation behaviours even if the algorithms started with a 2-approximation. This shows that edge-based representation does not in general help with maintaining 2-approximations of the vertex cover problem, if the simple fitness function of the node-based approach is used. For the third approach in which the edge-based representation is used together with a specific fitness function, we have shown that 2-approximations are maintained easily by recomputing maximal matchings of the dynamically changing graph.

The edge-based representation for this problem is inspired by the matching problem, which is the dual problem of the vertex cover problem. The concept of duality is used in Chapter 9 in a more general form to introduce simple randomised algorithms with a multi-valued edge-based encoding that finds 2-approximations for the weighted vertex cover problem. We have proven pseudo-polynomial upper bounds for RLS and the

(1+1) EA until they have achieved a 2-approximation. Furthermore, we have investigated the use of step-size adaptation in both algorithms and found that this technique improves the upper bound of RLS to polynomial time; while a pseudo-polynomial lower bound is still found for the (1+1) EA.

# Bibliography


[1] F. N. Abu-Khzam, R. L. Collins, M. R. Fellows, M. A. Langston, W. H. Suters, and C. T. Symons. Kernelization algorithms for the vertex cover problem: Theory and experiments. *ALENEX/ANALC*, 69, 2004.

[2] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovász, and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science (FOCS '79)*, pages 218–223. IEEE Press, 1979.

[3] D. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. Concorde tsp solver. *http://www.math.uwaterloo.ca/tsp/concorde/index.html*, 2013.

[4] D. Applegate, W. Cook, S. Dash, and A. Rohe. Solution of a min-max vehicle routing problem. *INFORMS J. on Computing*, 14(2):132–143, Apr. 2002.

[5] D. L. Applegate, R. E. Bixby, and V. Chvátal. *Princeton Series in Applied Mathematics : The Traveling Salesman Problem : A Computational Study*. Princeton University Press, Princeton, US, 2011.

[6] A. Auger and B. Doerr. *Theory of Randomized Search Heuristics: Foundations and Recent Developments*. World Scientific Publishing Co., Inc., 2011.

[7] M. Balinski. On the maximum matching, minimum covering. In *Proc. Symp. Math. Programming*, pages 434–445. Princeton University Press, 1970.

[8] N. Bansal and S. Khot. Inapproximability of hypergraph vertex cover and applications to scheduling problems. In S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, and P. Spirakis, editors, *Automata, Languages and Programming*, volume 6198 of *Lecture Notes in Computer Science*, pages 250–261. Springer Berlin Heidelberg, 2010.

[9] R. Bar-Yehuda and S. Even. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2(2):198 – 203, 1981.

[10] R. Bellman. Combinatorial processes and dynamic programming. r. bellman, m. hall, jr., eds. *Combinatorial Analysis*, page 217– 249, 1960.

[11] R. Bellman. Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1):61–63, Jan. 1962.

[12] H. Beyer and H. Schwefel. Evolution strategies - A comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

[13] S. Böttcher, B. Doerr, and F. Neumann. Optimal fixed and adaptive mutation rates for the leadingones problem. In *Conference on Problem Solving from Nature (PPSN)*, pages 1–10, 2010.

[14] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

[15] K. L. Clarkson. A modification of the greedy algorithm for vertex cover. *Inf. Process. Lett.*, 16(1):23–25, 1983.

[16] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, US, 2009.

[17] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[18] D. Corus, P. K. Lehre, and F. Neumann. The generalized minimum spanning tree problem: a parameterized complexity analysis of bi-level optimisation. In C. Blum and E. Alba, editors, *GECCO*, pages 519–526. ACM, 2013.

[19] D. Corus, P. K. Lehre, F. Neumann, and M. Pourhassan. A parameterized complexity analysis of bi-level optimisation with evolutionary algorithms. *Evolutionary Computation (to appear), doi: 10.1162/EVCO_a_00147*, 2015.

[20] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.

[21] D.-C. Dang and P. K. Lehre. Self-adaptation of mutation rates in non-elitist populations. In *Conference on Problem Solving from Nature (PPSN)*, 2016.

[22] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954.

[23] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008.

[24] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimisation: Nsga-ii. In *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, PPSN VI, pages 849–858, London, UK, UK, 2000. Springer-Verlag.

[25] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Proceedings of the 6th International Conference of Parallel Problem Solving from Nature: PPSN VI*, pages 849–858. Springer Berlin Heidelberg, 2000.

[26] K. Deb and A. Sinha. Solving bilevel multi-objective optimization problems using evolutionary algorithms. In M. Ehrgott, C. M. Fonseca, X. Gandibleux, J.-K. Hao, and M. Sevaux, editors, *EMO*, volume 5467 of *Lecture Notes in Computer Science*, pages 110–124. Springer, 2009.

[27] K. Deb and A. 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.

[28] V. G. Deineko, M. Hoffmann, Y. Okamoto, and G. J. Woeginger. The traveling salesman problem with few inner points. *Oper. Res. Lett.*, 34(1):106–110, 2006.

[29] B. Doerr and C. Doerr. Optimal parameter choices through self-adjustment: Applying the 1/5-th rule in discrete settings. In *Genetic and Evolutionary Computation Conference (GECCO)*, pages 1335–1342, 2015.

[30] B. Doerr, C. Doerr, and F. Ebel. From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science*, 567:87 – 104, 2015.

[31] B. Doerr, C. Doerr, and T. Kötzing. The right mutation strength for multi-valued decision variables. In *Genetic and Evolutionary Computation Conference (GECCO)*, 2016.

[32] B. Doerr, C. Doerr, and J. Yang. k-bit mutation with self-adjusting k outperforms standard bit mutation. In *Parallel Problem Solving from Nature – PPSN XIV: 14th International Conference, Edinburgh, UK, September 17-21, 2016, Proceedings*, pages 824–834. Springer International Publishing, 2016.

[33] B. Doerr, D. Johannsen, and M. Schmidt. Runtime analysis of the (1+1) evolutionary algorithm on strings over finite alphabets. In *Workshop on Foundations of Genetic Algorithms (FOGA)*, pages 119–126, 2011.

[34] B. Doerr, D. Johannsen, and C. Winzen. Multiplicative drift analysis. *Algorithmica*, 64(4):673–697, 2012.

[35] B. Doerr and S. Pohl. Run-time analysis of the (1+1) evolutionary algorithm optimizing linear functions over a finite alphabet. In *Genetic and Evolutionary Computation Conference (GECCO)*, pages 1317–1324, 2012.

[36] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999. 530 pp.

[37] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Springer Publishing Company, Incorporated, 2013.

[38] S. Droste. Analysis of the (1+1) EA for a dynamically changing ONEMAX-variant. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 1, pages 55–60, May 2002.

[39] D.-Z. Du, K.-I. Ko, and X. Hu. *Design and Analysis of Approximation Algorithms*. Springer Publishing Company, Incorporated, 2011.

[40] A. E. Eiben, Z. Michalewicz, M. Schoenauer, and J. E. Smith. Parameter control in evolutionary algorithms. In *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 19–46. Springer, 2007.

[41] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. SpringerVerlag, 2003.

[42] M. Englert, H. Röglin, and B. Vöcking. Worst case and probabilistic analysis of the 2-opt algorithm for the TSP. *Algorithmica*, 68(1):190–264, 2014.

[43] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. Wiley, 3rd edition, 1968.

[44] M. Fischetti, J. J. Salazar González, and P. Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45(3):378–394, 1997.

[45] J. Flum and M. Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[46] C. M. Fonseca and P. J. Fleming. Genetic algorithms for multiobjective optimization: Formulationdiscussion and generalization. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 416–423, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[47] T. Friedrich, J. He, N. Hebbinghaus, F. Neumann, and C. Witt. Analyses of simple hybrid algorithms for the vertex cover problem. *Evolutionary Computation*, 17(1):3–19, 2009.

[48] T. Friedrich, J. He, N. Hebbinghaus, F. Neumann, and C. Witt. Approximating covering problems by randomized search heuristics using multi-objective models. *Evolutionary Computation*, 18(4):617–633, 2010.

[49] M. Gendreau and J.-Y. Potvin. *Handbook of Metaheuristics*. Springer Publishing Company, Incorporated, 2nd edition, 2010.

[50] R. H. Gonzales. Solution to the traveling salesman problem by dynamic programming on the hypercube. Technical report, Technical Report Number 18, Operations Research Center, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, 1962.

[51] S. Guha, R. Hassin, S. Khuller, and E. Or. Capacitated vertex covering. *JOURNAL OF ALGORITHMS*, 48:257–270, 2003.

[52] G. Gutin and D. Karapetyan. A memetic algorithm for the generalized traveling salesman problem. *Natural Computing*, 9(1):47–60, 2010.

[53] B. Hajek. Hitting-time and occupation-time bounds implied by drift analysis with applications. *Advances in Applied Probability*, 13(3):502–525, 1982.

[54] N. Hansen, S. D. Müller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

[55] P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In *Meta-heuristics*, pages 433–458. Springer US, 1999.

[56] E. Happ, D. Johannsen, C. Klein, and F. Neumann. Rigorous analyses of fitness-proportional selection for optimizing linear functions. In *Conference on Genetic and Evolutionary Computation (GECCO)*, pages 953–960, 2008.

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

[58] M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. In *Proceedings of the 1961 16th ACM National Meeting*, ACM '61, pages 71.201–71.204, New York, NY, USA, 1961. ACM.

[59] M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.

[60] M. Held and R. M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

[61] M. Held and R. M. Karp. The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical Programming*, 1(1):6–25, 1971.

[62] D. S. Hochbaum. Efficient bounds for the stable set, vertex cover and set packing problems. *Discrete Applied Mathematics*, 6(3):243 – 254, 1983.

[63] S. Hong. *A Linear Programming Approach for the Traveling Salesman Problem*. PhD thesis, Johns Hopkins University, Baltimore, Maryland, USA, 1972.

[64] C. Horoba and F. Neumann. Benefits and drawbacks for the use of $\epsilon$-dominance in evolutionary multi-objective optimization. In *In Proc. of GECCO 2008*, 2008.

[65] B. Hu and G. Raidl. An evolutionary algorithm with solution archives and bounding extension for the generalized minimum spanning tree problem. In *Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference*, GECCO '12, pages 393–400, New York, NY, USA, 2012. ACM.

[66] B. Hu and G. R. Raidl. Effective neighborhood structures for the generalized traveling salesman problem. In J. I. van Hemert and C. Cotta, editors, *EvoCOP*, volume 4972 of *Lecture Notes in Computer Science*, pages 36–47. Springer, 2008.

[67] Z. Ivković and E. Lloyd. Fully dynamic maintenance of vertex cover. In J. van Leeuwen, editor, *Graph-Theoretic Concepts in Computer Science*, volume 790 of *Lecture Notes in Computer Science*, pages 99–111. Springer Berlin Heidelberg, 1994.

[68] A. H. G. R. K. J. K. Lenstra. Some simple applications of the travelling salesman problem. *Operational Research Quarterly (1970-1977)*, 26(4):717–733, 1975.

[69] T. Jansen. *Analyzing Evolutionary Algorithms - The Computer Science Perspective*. Natural Computing Series. Springer, 2013.

[70] T. Jansen, P. S. Oliveto, and C. Zarges. Approximating vertex cover using edge-based representations. In F. Neumann and K. A. D. Jong, editors, *Foundations of Genetic Algorithms XII, FOGA '13, Adelaide, SA, Australia, January 16-20, 2013*, pages 87–96. ACM, 2013.

[71] D. Karapetyan and G. Gutin. Efficient local search algorithms for known and new neighborhoods for the generalized traveling salesman problem. *European Journal of Operational Research*, 219(2):234–251, 2012.

[72] A. Koh. Solving transportation bi-level programs with differential evolution. In *IEEE Congress on Evolutionary Computation*, pages 2243–2250. IEEE, 2007.

[73] T. Kötzing, A. Lissovoi, and C. Witt. (1+1) ea on generalized dynamic onemax. In *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII*, FOGA '15, pages 40–51, New York, NY, USA, 2015. ACM.

[74] T. Kötzing, A. Lissovoi, and C. Witt. (1+1) EA on generalized dynamic onemax. In *Workshop on Foundations of Genetic Algorithms (FOGA)*, pages 40–51, 2015.

[75] T. Kötzing, F. Neumann, H. Röglin, and C. Witt. Theoretical analysis of two aco approaches for the traveling salesman problem. *Swarm Intelligence*, 6(1):1–21, 2012.

[76] S. Kratsch, P. K. Lehre, F. Neumann, and P. S. Oliveto. Fixed parameter evolutionary algorithms and maximum leaf spanning trees: A matter of mutation. In R. Schaefer, C. Cotta, J. Kolodziej, and G. Rudolph, editors, *Parallel Problem Solving from Nature - PPSN XI, 11th International Conference, Kraków, Poland, September 11-15, 2010, Proceedings, Part I*, volume 6238 of *Lecture Notes in Computer Science*, pages 204–213. Springer, 2010.

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

[78] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler. Combining convergence and diversity in evolutionary multiobjective optimization. *Evol. Comput.*, 10(3):263–282, Sept. 2002.

[79] F. Legillon, A. Liefooghe, and E.-G. Talbi. Cobra: A cooperative coevolutionary algorithm for bi-level optimization. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2012.

[80] P. K. Lehre and C. Witt. General drift analysis with tail bounds. Technical report, http://arxiv.org/abs/1307.2559, 2013.

[81] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Oper. Res.*, 21(2):498–516, Apr. 1973.

[82] J. D. C. Little, K. G. Murty, D. W. Sweeney, and C. Karel. An algorithm for the traveling salesman problem. *Oper. Res.*, 11(6):972–989, Dec. 1963.

[83] J. Lässig and D. Sudholt. Adaptive population models for offspring populations and parallel evolutionary algorithms. Technical report, https://arxiv.org/abs/1102.0588, 2011.

[84] P. Miliotis. Integer programming approaches to the travelling salesman problem. *Mathematical Programming*, 10(1):367–378, 1976.

[85] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995.

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

[87] S. Nallaperuma. *Parameterized Analysis of Bio-inspired Computation and the Traveling Salesperson Problem*. PhD thesis, School of Computer Science, University of Adelaide, 2015.

[88] S. Nallaperuma, A. M. Sutton, and F. Neumann. Fixed-parameter evolutionary algorithms for the euclidean traveling salesperson problem. In *IEEE Congress on Evolutionary Computation*, pages 2037–2044. IEEE, 2013.

[89] F. Neumann and J. Reichel. Approximating minimum multicuts by evolutionary multi-objective algorithms. In G. Rudolph, T. Jansen, S. M. Lucas, C. Poloni, and N. Beume, editors, *Parallel Problem Solving from Nature - PPSN X, 10th International Conference Dortmund, Germany, September 13-17, 2008, Proceedings*, volume 5199 of *Lecture Notes in Computer Science*, pages 72–81. Springer, 2008.

[90] F. Neumann, J. Reichel, and M. Skutella. Computing minimum cuts by randomized search heuristics. *Algorithmica*, 59(3):323–342, 2011.

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

[92] R. O'Callahan and J.-D. Choi. Hybrid dynamic data race detection. In *Proceedings of the Ninth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '03, pages 167–178, New York, NY, USA, 2003. ACM.

[93] P. Oliveto and C. Witt. Simplified drift analysis for proving lower bounds in evolutionary computation. *Algorithmica*, 59(3):369–386, 2011.

[94] P. Oliveto and C. Witt. Erratum: Simplified drift analysis for proving lower bounds in evolutionary computation. Technical report, http://arxiv.org/abs/1211.7184, 2012.

[95] P. S. Oliveto, J. He, and X. Yao. Analysis of the (1+1)-EA for finding approximate solutions to vertex cover problems. *IEEE Trans. Evolutionary Computation*, 13(5):1006–1029, 2009.

[96] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Rev.*, 33(1):60–100, Feb. 1991.

[97] S. Pirzada and A. Dharwadker. Applications of graph theory. *Journal of Korean society for Inductrial and applied mathematics*, 11(4):19–38, 2007.

[98] P. C. Pop and S. Iordache. A hybrid heuristic approach for solving the generalized traveling salesman problem. In N. Krasnogor and P. L. Lanzi, editors, *GECCO*, pages 481–488. ACM, 2011.

[99] M. Pourhassan, T. Friedrich, and F. Neumann. On the use of the dual formulation for minimum vertex cover in evolutionary algorithms. In *Proceedings of the 14th Conference on Foundations of Genetic Algorithms (FOGA'17)*. ACM, submitted.

[100] M. Pourhassan, W. Gao, and F. Neumann. Maintaining 2-approximations for the dynamic vertex cover problem using evolutionary algorithms. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, pages 903–910, New York, NY, USA, 2015. ACM.

[101] M. Pourhassan and F. Neumann. On the impact of local search operators and variable neighbourhood search for the generalized travelling salesperson problem. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*, GECCO '15, pages 465–472, New York, NY, USA, 2015. ACM.

[102] M. Pourhassan and F. Neumann. Theoretical analysis of local search and simple evolutionary algorithms for the generalized travelling salesperson problem. *Evolutionary Computation*, submitted.

[103] M. Pourhassan, F. Shi, and F. Neumann. Parameterized analysis of multi-objective evolutionary algorithms and the weighted vertex cover problem. *Evolutionary Computation*, submitted.

[104] M. Pourhassan, F. Shi, and F. Neumann. Parameterized analysis of multi-objective evolutionary algorithms and the weighted vertex cover problem. In *Proceedings of the 14th Conference on Parallel Problem Solving from Nature (PPSN 2016)*. Springer, to appear.

[105] L. V. Quintas and F. Supnick. On some properties of shortest hamiltonian circuits. *The American Mathematical Monthly*, 72(9):977–980, 1965.

[106] J. D. Schaffer. *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms*. PhD thesis, Vanderbilt University, Tennessee, USA, 1984.

[107] N. Srinivas and K. Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evol. Comput.*, 2(3):221–248, Sept. 1994.

[108] S. A. Stanhope and J. M. Daida. Genetic algorithm fitness dynamics in a changing environment. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC1999)*, pages 1851–1858, Piscataway, NJ, 1999. IEEE.

[109] D. Sudholt. Hybridizing evolutionary algorithms with variable-depth search to overcome local optima. *Algorithmica*, 59(3):343–368, 2011.

[110] D. Sudholt. Parametrization and balancing local and global search. In F. Neri, C. Cotta, and P. Moscato, editors, *Handbook of Memetic Algorithms*, volume 379 of *Studies in Computational Intelligence*, pages 55–72. Springer, 2012.

[111] A. M. Sutton and F. Neumann. A parameterized runtime analysis of evolutionary algorithms for the Euclidean traveling salesperson problem. In *Proceedings of the Twenty-Sixth Conference on Artificial Intelligence (AAAI-12)*, 2012.

[112] A. M. Sutton and F. Neumann. A parameterized runtime analysis of simple evolutionary algorithms for makespan scheduling. In *Proceedings of the Twelfth Conference on Parallel Problem Solving from Nature (PPSN 2012)*, pages 52–61. Springer, 2012.

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

[114] M. Theile. Exact solutions to the traveling salesperson problem by a population-based evolutionary algorithm. In C. Cotta and P. Cowling, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 5482 of *Lecture Notes in Computer Science*, pages 145–155. Springer Berlin Heidelberg, 2009.

[115] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[116] I. Wegener. *Methods for the Analysis of Evolutionary Algorithms on Pseudo-Boolean Functions*, pages 349–369. Springer US, Boston, MA, 2002.