UNIVERSITY OF ADELAIDE

DOCTORAL THESIS

Bio-Inspired Computing For Complex And Dynamic Constrained Problems

Author:
Vahid ROOSTAPOUR

Supervisor:
Prof. Frank NEUMANN

Co-Supervisors:
Dr. Wanru GAO
Dr. Mojgan POURHASSAN

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

in the

Optimisation and Logistics
School of Computer Science

October, 2020

Declaration of Authorship

I, Vahid ROOSTAPOUR, 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 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: October 7, 2020

UNIVERSITY OF ADELAIDE

Abstract

Faculty of Engineering, Computer and Mathematical Sciences

School of Computer Science

Doctor of Philosophy

Bio-Inspired Computing For Complex And Dynamic Constrained Problems

by Vahid ROOSTAPOUR

Bio-inspired algorithms are general-purpose optimisation methods that can find solutions with high qualities for complex problems. They are able to find these solutions with minimal knowledge of a search space. Bio-inspired algorithms (the design of which is inspired by nature) can easily adapt to changing environments. In this thesis, we contribute to the theoretical and empirical understanding of bio-inspired algorithms, such as evolutionary algorithms and ant colony optimisation. We address complex problems as well as problems with dynamically changing constraints. Firstly, we review the most recent achievements in the theoretical analysis of dynamic optimisation via bio-inspired algorithms. We then continue our investigations in two major areas: static and dynamic combinatorial problems. To tackle static problems, we study the evolutionary algorithms that are enhanced by using a knowledge-based mutation approach in solving single- and multi-objective minimum spanning tree (MST) problems. Our results show that proper development of biased mutation can significantly improve the performance of evolutionary algorithms. Afterwards, we analyse the ability of single- and multi-objective algorithms to solve the packing while travelling (PWT) problem. This $\mathcal{NP}$-hard problem is chosen to represent real-world multi-component problems. We outline the limitations of randomised local search in solving PWT and prove the advantage of using evolutionary algorithms. Our dynamic investigations begin with an empirical analysis of the ability of simple and advanced evolutionary algorithms to optimise the dynamic knapsack (KP) problem. We show that while optimising a population of solutions can speed up the ability of an algorithm to find optimal solutions after a dynamic change, it has the exact opposite effect in environments with high-frequency changes. Finally, we investigate the dynamic version of a more general problem known as the subset selection problem. We prove the inability of the adaptive greedy approach to maintain quality solutions in dynamic environments and illustrate the advantage of using evolutionary algorithms theoretically and practically.

Acknowledgements

I would like to express my sincere appreciation to my principal supervisor, Prof. Frank Neumann, for his patience, motivation and immense knowledge, and for his continuous support of my PhD study. It's been an honour to be one of his PhD students. I thank him for encouraging my research and for allowing me to grow as a research scientist.

I would like to extend my sincere thanks to my co-supervisors, Dr Mojgan Pourhassan and Dr Wanru Gao, for all their support and priceless guidance during my PhD.

Last but not least, I gratefully thank all of the co-authors of the papers built on the research in this thesis: Aneta Neumann, Jakob Bossek, and Tobias Friedrich. I learned a lot in collaboration with them.

Contents

List of Figures

List of Tables

To Mitra

Chapter 1

Introduction

Bio-inspired computing methods are well-known general problem-solvers that are applicable to various problems. These algorithms (which include evolutionary algorithms (EA) [ES15], ant colony optimisation(ACO) [DS04], and particle swarm optimisation (PSO) [KE95]) are inspired by procedures that occur in nature. EAs, for example, mimic principles from Darwinian evolutionary theory, and ACO is inspired by the pheromone-based communication of biological ants, which serves to find the nearest path between the food source and the nest. These algorithms use random operators and do not require much advance knowledge about the problem under consideration. They are able to achieve high-quality solutions in cases where the complexity of a problem makes problem-specific algorithms difficult to find. During the last two decades, these algorithms have been successfully implanted in a vast range of applications, such as combinatorial optimisation problems [NW10b; SK18; CWM11].

Combinatorial optimisation problems are the set of problems over finite discrete search spaces that are bounded by a set of constraints, and the goal is to find the optimal solution(s) with respect to given evaluation function(s). These problems include a variety of theoretical and real-world applications with different levels of computational complexity. The minimum spanning tree (MST) problem, knapsack (KP) problem, and packing while travelling (PWT) problem are well-known combinatorial optimisation problems that represent real-world applications. While some of the combinatorial optimisation problems, such as MST, are solvable in polynomial time, many others are proven to be $\mathcal{NP}$-hard. The complexity of $\mathcal{NP}$-hard problems, such as KP and PWT, is also different. PWT is proven not to have an algorithm that guarantees a constant approximation ratio in polynomial time unless $\mathcal{P} = \mathcal{NP}$, while there is a 2-approximation algorithm for KP (see Chapter 3 for the details). Due to the ability of bio-inspired algorithms to explore the search space of complex problems, their performance in optimising combinatorial problems has been studied extensively in the literature.

1.1 Contibutions and Background

A wide range of studies have investigated the performance of bio-inspired algorithms in practice, since these algorithms can easily be applied to a variety of problems. Moreover, the key feature of these algorithms is their use of random operators to explore the search space, and theoretical analysis of random processes is challenging. Thus, although significant improvements have been achieved during the last two decades in the theoretical understanding of bio-inspired strategies, there is still a considerable gap between the theoretical and practical understanding of these algorithms. This gap in understanding needs to be addressed [DN20; Jan13; Pop14]. Theoretical investigations of bio-inspired methods have mostly considered the performance of evolutionary algorithms and ant colony optimisation algorithms. These investigations have included studies of polynomially solvable problems (such as MST and the shortest path problem) as well as well-known $\mathcal{NP}$-hard problems, such as KP and subset selection [MA94; NW10a; QBF20; STW04].

The complexity of many combinatorial problems comes from their objective functions. However, in real-world applications, another challenge is to deal with dynamic changes in the constraints during the optimisation process. Evolutionary algorithms (in their capacity of general-purpose problem-solving methods) have been widely used to tackle dynamic optimisation problems, both theoretically and empirically [Yan15]. Many theoretical investigations have considered the performance of evolutionary algorithms in dynamic environments, where either the constraint(s) or some properties of the search space might change during the optimisation process. Neumann and Witt studied the behaviour of the simplest evolutionary algorithm ($(1+1)$ EA) on the dynamic makespan scheduling problem, in which the adversary can change the processing time of a job [NW15]. They calculated the expected time of finding a good-quality solution in different situations, where the algorithms either started from scratch and the dynamic changes happened frequently, or the algorithms started from a good-quality solution and aimed to find a new good-quality solution after one change. The same analyses have also been applied to dynamic graph problems where a dynamic change might add/remove an edge to/from a graph [Bos+19; PRN19].

In this thesis, we review the most recent theoretical results relating to the performance of evolutionary algorithms and ant colony optimisation in the context of problems with dynamically changing constraints. We discuss how dynamism has been applied to pseudo-Boolean benchmark functions and the specific case of the ONEMAX problem, in addition to some well-known combinatorial optimisation problems such as the vertex cover problem, makespan scheduling, and the Maze problem. We describe the application of theoretical tools to analysing the performance of evolutionary algorithms and ant colony optimisation.

After completing a review of the most recent theoretical results, we study the benefits of using a knowledge-based mutation operator in evolutionary algorithms to solve the single- and multi-objective MST (moMST) problems. The main idea is to change the direction of the search based on external knowledge about the features of the minimum spanning tree. For the classical MST problem, it is legitimate to assume that edges of low weight/rank are more likely to be in an MST than edges of high weight/rank. Such knowledge can also be leveraged in terms of biased mutation, as demonstrated by Raidl et al. [RKJ06] on random graphs. The authors showed that mutation, where the edge selection probability is biased towards lower rank edges, can lead to immense speedups for evolutionary algorithms for different sub-graph selection problems, inter alia the MST. Likewise, for the moMST problem, non-dominated spanning trees are more likely to be composed of edges which are dominated by few other edges, i.e., edges of low non-domination level or domination number. A recent study by Bossek et al. [BGN19] confirms this assumption empirically. Both Raidl et al. and Bossek et al. consider the simple edge-exchange mutation on spanning trees: an edge is added to a spanning tree and an edge is dropped from the unique introduced cycle to obtain another spanning tree. Bias is introduced by modifying the edge selection probability to favour low-rank edges. We show that although the biased mutation can significantly improve the running time of evolutionary algorithms in terms of finding the MST, it might require exponential expected running time when heavy edges are frequent and essential in the MST. To avoid the exponential time, as well as to exploit the advantages of using biased mutation, we introduce a mixed mutation operator. This operator performs the biased mutation in half of the iterations and uses the common uniform mutation in the other half. We prove that this new mutation operator improves the state-of-the-art results for finding MST via evolutionary algorithms.

We continue our theoretical investigation of static problems by analysing the packing while travelling (PWT) problem [PN17]. PWT, also known as the nonlinear knapsack problem, is a modified version of a more general problem called the travelling thief problem (TTP) [BMB13]. These problems, which are common in real-world applications and known as multi-components, include more than one component to be solved [Bon+19]. Their final objective function integrates the components in a way that optimising a single component does not necessarily result in a final high-quality solution. The TTP problem, for example, is a combination of the travelling salesman problem (TSP) and KP. In TTP, there is a graph of connected cities, and each city has a number of items with positive profits and weights. A vehicle with constant capacity has to visit each city exactly once and pick up the items in a way that maximises the benefit function. As the vehicle packs an item, the item's weight impacts the velocity of the vehicle. The higher the weight of the item, the lower the velocity of the vehicle. The final benefit function combines the profit of the packed items and the cost function, which is calculated as the total time that the vehicle takes to traverse the chosen path according to the weight of the packed items. The cost function

in this problem is nonlinear. In PWT, the problem is only characterised by a fixed route. However, it is proven that PWT is $\mathcal{NP}$-hard, and there is no approximation algorithm with a constant approximation ratio, even for two cities [PN17]. While the objective function is still nonlinear, we consider a version of this problem in which the weights and profits of the items are highly correlated. We show that the classic randomised local search (RLS) algorithm is unable to escape from local optima and present a modified mutation for RLS to address this stucking issue. We prove an upper bound of $O(n^3)$ for our algorithm as well as the same upper bound for a well-known multi-objective algorithm called G-SEMO. The other instance of PWT that is considered in this thesis assumes items with weights equal to one. We show that the simple evolutionary algorithm, $(1+1)$ EA, finds the optimal solution of the uniform version in expected time $O(n^2 \log(\max\{n, p_{\max}\})$, where $p_{\max}$ is the maximum profit of the items. Finally, we compare the algorithms in practice and show that incorporating population can help local search algorithms to find the global optima.

Next, we analyse the performance of the well-known single- and multi-objective bio-inspired optimisation algorithms on two dynamic combinatorial problems. Most studies for dynamic problems so far focus on dynamic fitness functions [NYB12]. However, in real-world applications the optimisation goal, such as maximising profit or minimising costs, often does not change. Instead, resources to achieve this goal change over time and influence the quality of solutions that can be obtained. In the context of continuous optimisation, dynamically changing constraints have been investigated in [AHN18; NY12].

The empirical investigation of dynamic KP is motivated by theoretical studies in [Shi+19]. Shi et al. showed the efficacy of using multi-objective evolutionary algorithms in optimising linear functions under dynamic uniform constraints. They proved that a simple population-based algorithm outperforms $(1+1)$ EA with at least the factor $O(n)$. In terms of the dynamic setting, they assumed that the algorithms started with an optimal solution, right before a dynamic change altered the capacity. In the first part of our experiments, we examine their results by considering a version of KP in which the weights are set to one. We use the benchmarks provided in [Pol+14], which were initially created for TTP and include three different types of items in terms of the correlation between the weights and the profits with varying complexity. The dynamism in our benchmark has been applied by changing the capacity of knapsack, which is the constraint in KP, during the optimisation process. We study changes with different magnitudes, either from uniform or normal distributions, and different frequencies. While the initial experiments on KP with uniform weights confirm the results in [Shi+19], the examinations on the general KP show the importance of small details in the algorithm. Our results demonstrate the impact of the size of the population in multi-objective algorithms as well as the definition of dominance, which is crucial in using multi-objective algorithms. Moreover, we show that when dynamic changes happen with high frequency, the population

slows down the ability of the algorithms to track an optimal solution. The significant performance of G-SEMO as a baseline evolutionary algorithm motivates us to compare it with two well-known advanced algorithms in the literature, NSGA-II and SPEA2. We show that the emphasis of these advanced algorithms on the distribution of final solutions reduces their ability to track the optimal solution after dynamic changes.

We continue our studies on dynamic optimisation by considering the dynamic version of the monotone subset selection problem under monotone dynamic constraints. In this problem, which is the general version of all the combinatorial optimisation problems that have been studied in this thesis, a set of elements, an objective function, a cost function and a budget constraint are given. The goal is to select a subset of items such that it maximises the objective function without exceeding the budget constraint. To consider the general monotone functions, we use a specific property of set functions called submodularity. A set function f is, informally, submodular if, for sets $X \subseteq Y$, the contribution of adding a single element e to X is larger than the contribution of adding e to Y, i. e. $f(X \cup \{e\}) \geq f(Y \cup \{e\})$. Submodular functions and subset selection are widely used in artificial intelligence applications. Although the subset selection is $\mathcal{NP}$-hard, it has been proven that greedy algorithms guarantee the best possible worst-case approximation for monotone submodular functions. These results have been extended to monotone functions concerning an external factor called submodularity ratio α_f, which measures the submodularity of a given function. The performance of evolutionary algorithms on subset selection has been recently studied in [Bia+20] and [Qia+17]. They showed not only that multi-objective evolutionary algorithms could guarantee the best possible worst-case approximation similar to the generalised greedy algorithm, but that such algorithms could also find better solutions in real applications. Motivated by these studies, we investigate the performance of an adaptive version of the generalised greedy algorithm and G-SEMO on the monotone subset selection problem. They theoretically proved that the adaptive greedy algorithm performs arbitrarily bad facing the dynamic problem. On the other hand, starting with a zero solution and budget B, G-SEMO efficiently finds the optimal solution to all possible budgets smaller than B. Moreover, it efficiently repairs the population after a dynamic change occurs. Similarly to the empirical analysis of dynamic KP, we compare the performance of G-SEMO with NSGA-II and SPEA2 (which are the established multi-objective algorithms in the literature) in optimising the dynamic version of two different real-world benchmarks. Our results demonstrate that, for some constraints, the greedy algorithm outperforms evolutionary algorithms at the early stages of the optimisation. However, as the process continues, the evolutionary algorithms maintain their population in a way that allows them to respond adequately to the dynamic changes. We also show that if there is enough time to react after a dynamic change, there is no incremental improvement to be gained by using NSGA-II and SPEA-II in comparison with G-SEMO.

1.2 Thesis Organisation

This thesis is structured as follows. In Part I, which relates to the background, Chapter 2 introduces the bio-inspired algorithms and the analytical methods that are used in this thesis. In Chapter 3, a brief history and the exact definitions of the problems that we analyse are presented. Chapter 4 is a survey that reviews the most recent results of evolutionary algorithms and ant colony optimisation on well-known problems. Part II of this thesis presents the results of our investigations on static problems. In Chapter 5, we outline our studies on the impact of using biased mutation in evolutionary algorithms to solve single- and multi-objective MSTs. The results of analysing the baseline evolutionary algorithms facing PWT are explained in Chapter 6. In Part III of the thesis, we present our investigation on the behaviour of evolutionary algorithms optimising combinatorial problems with dynamically changing constraints. We empirically analyse the performance of well-known bio-inspired algorithms on the dynamic KP in Chapter 7. In Chapter 8, we consider a more general problem in this field and show the outperformance of evolutionary algorithms in finding high-quality solutions for the subset selection problem in comparison with greedy algorithms. Finally, Chapter 9 concludes the thesis and presents the highlights of the study.

Part I

Basics

Chapter 2

Bio-Inspired Computing and Analytical Methods

2.1 Introduction

Bio-inspired computing incorporates computing methods inspired by nature. These algorithms iteratively search for better solutions and employ *heuristics* that involve randomness. They are also part of a broader class of *stochastic search algorithms*. The random behaviour of bio-inspired computing methods means that they can be considered general problem-solving techniques, capable of finding high-quality solutions for $\mathcal{NP}$-hard problems.

Generally, the optimisation algorithms that we consider in this thesis begin the optimisation process using the initialised solution x. Defining the *fitness function* $f : D \to R$, the algorithms explore the *search space* D to find solution y such that $f(y)$ will be better than $f(x)$. They repeat the search procedure iteratively, with the aim of finding a solution x^* with optimal fitness value.

While stochastic search algorithms have been developed rapidly, the theoretical analysis of their behaviour is still far behind is still far behind their practical implementation. The main reason for this is the lack of theoretical tools and the high complexity of analysing random decisions that are made during the optimisation process. However, there have been significant achievements in developing tools and techniques to study the randomness of bio-inspired algorithms during the past decade.

In this chapter, we introduce the bio-inspired algorithms and analytical methods that are used in this thesis. In order to provide a detailed explanation , we use a well-known pseudo-Boolean benchmark function called LEADINGONES. In this problem, solutions $x = (x_1, \cdots, x_n) \in \{0,1\}^n$ are binary bit-strings of size n. The fitness value of a solution is the length of consecutive 1 bits that start from the first position. The fitness function is mathematically defined as $LO(x) = \sum_{j=1}^{n} \prod_{i=1}^{j} x_i$.

This chapter is structured in two parts. The first part begins by introducing local search and the randomised local search algorithm (Section 2.2). Next, in Section 2.3,

Algorithm 1: Local Search

```
1 The initial solution x and neighbourhood function 𝒩(x) are given;
2 while stopping criterion not met do
3     Replace x with the better solution y ∈ 𝒩(x)
```

we present the main concepts of evolutionary algorithms, how to model a problem in order to solve it using evolutionary algorithms, and the single- and multi-objective algorithms that are considered in this thesis. In the second part of this chapter, we introduce the analytical methods which are commonly used to analyse the performance of bio-inspired algorithms. We present deviation bounds in Section 2.4.1 , followed by recent approaches such as fitness-based partitioning and drift analysis (Sections 2.4.2 and 2.4.3, respectively).

2.2 Local Search

As the name suggests, local search algorithms try to find a good-quality solution by locally searching a portion of the search space. A local search algorithm starts from an initial solution (chosen randomly or initialised using another algorithm) and iteratively explores its *neighbourhood*for a better quality solution (see Algorithm 1). Note that an appropriate neighbourhood function is required to define the neighbourhood. When a new solution is found, the search continues, using the new solution as the initial point. The search stops when the algorithm is unable to improve the solution, which occurs when it has either found the optimal solution or become stuck in local optima.

In complex search spaces, it is common that local search algorithms become stuck in a local optimum point. In other words, they find a solution that is better than its surroundings, but which is not necessarily the global optima. To avoid this problem, a proper definition of neighbourhood function $\mathcal{N}(x)$ is necessary. The neighbourhood is mainly chosen based on the solution representation. For bit-string solutions, it is common to define the neighbourhood using the Hamming distance. For two solutions $x, y \in \{0,1\}^n$, Hamming distance $H(x,y) = \sum_{i=1}^{n} = |x_i - y_i|$ counts the number of bits that have different values to each other. The neighbourhood of a bit-string solution x is defined by solutions y, such that $H(x,y)$ is either less than some constant value c or equal to c. When the solutions are represented by real values, one can use the Euclidean distance function to determine the neighbourhood. Size of neighbourhood significantly impacts the final results of local search algorithms. Searching a larger portion of the search space might prevent the algorithm from converging on an optimal point. On the other hand, searching a small neighbourhood causes fast convergence with insufficient exploration. One approach which benefits from this trade-off is to use a dynamic neighbourhood size. In such cases, the algorithms start with the larger neighbourhood, which entails *exploration*, and reduce it

Algorithm 2: RLS

```
1 The initial solution x is given;
2 while stopping criterion not met do
3     y ← flip one bit of x chosen uniformly at random;
4     if f(y) ≥ f(x) then
5         x ← y;
```

gradually, so that perform the *exploitation*.

Another considerable issue in local search algorithms is the direction of the search. This issue entails two main questions: what is the criteria for the algorithm to accept a new solution, and how does the algorithm choose the next solution? Regarding the quality of new solutions, to prevent the algorithm from becoming stuck at a local optimum point, it might be necessary for the algorithms to be able to accept solutions with at least the same quality as that of the current search point. In local search algorithms, there are two approaches to searching the neighbourhood. One is to explore all the possible neighbours of the current solution, and to then choose the best one , in terms of the fitness value, as the next point. This approach, however, does not apply to many problems, because of the size of the search space. This approach is impractical in settings where the search space is prohibitively large. The other approach is to select the first solution in the neighbourhood that satisfies the criteria.

The following section describes the randomised local search algorithm. This algorithm chooses a neighbour solution by performing a limited number of random changes to the current solution.

2.2.1 Randomised Local Search

Consider the LEADINGONES problem introduced in Section 2.1. Let x denote the initial solution. The simplest approach to maximising $LO(x)$ is to iteratively choose a random solution y near x and replace x with y if $LO(y) \geq LO(x)$. This is precisely what the randomised local search (RLS) does.

The progress continues until RLS cannot find a better solution within a preset number of iterations. Note that the size of neighbourhood plays an important role in this algorithm. In the coming chapters, we will present some The process continues until RLS cannot find a better solution within a pre-set number of iterations. Note that the size of the neighbourhood plays an important role in this algorithm. In the coming chapters, we will present some situations in which RLS cannot escape from local optima due to the small size of the neighbourhood. On the other hand, searching in a large area prevents the algorithm from converging on a high-quality solution.

In this thesis, we consider problems in which the solutions are presented as binary bit-strings with size n. For such problems, the standard RLS algorithm is presented

in Algorithm 2. Starting with an initial solution x, RLS creates a new solution y by flipping exactly one bit of x uniformly at random, and replacing x with y if the new solution is superior in terms of fitness. The algorithm repeats these steps for as long as the stopping criterion is not met.

The standard RLS uses the Hamming distance to determine the neighbourhood of the current solution. It searches among solutions that are different from the current solution in a single bit. While the fixed number of bit flips in this method eases the theoretical analysis of RLS, we will consider problems in this thesis that are proven to require more than one bit-flip to escape the local optima (Chapter 6). In such cases, we modify the bit-flip process in RLS.

2.3 Evolutionary Computation

Evolutionary algorithms (EAs) are randomised general-purpose problem-solving methods that mimic principles from Darwinian evolutionary theory based on the concepts of selection, inheritance, competence, survival and reproduction. These algorithms use a *population* of *individuals* and prioritise members of the population with higher fitness values as potential parents of the next generation. These algorithms have proven successful in a wide range of applications, in particular, in tackling combinatorial $\mathcal{NP}$-hard optimisation problems [CWM11; Deb01]. In Sections 2.3.1 to 2.3.5, we describe the four main issues to be addressed properly before performing an evolutionary algorithm on a given problem.

2.3.1 Solution Representation

While evolutionary algorithms can be applied to various problems, they are only capable of obtaining reliable results in the case of problems that have been appropriately modelled. The first step in using EAs is to select *solution representation*. Either a single real number, a binary value, a string, or a combination of different values can represent a solution. For example, if we need to optimise the location of a facility in a two-dimensional space, we can use an array of two real values as the solution representation. In the LEADINGONES example, each solution is represented using a sequence of zero/one bits.

2.3.2 Parent Selection

When performing evolutionary algorithms, the next step is to define the *selection* operation. From a population of individuals, the selection operator must choose parent solutions in order to produce the next *generation*. The selection must take into account the quality of solutions, allowing for the fact that solutions with better fitness value have a greater chance of becoming parents. In this way, the algorithm assures that the *offspring* inherit the beneficial properties of better solutions. On the other hand, it must not overcome the random nature of the evolutionary algorithms,

i.e., even the worst solutions must have some chance of becoming parents, since they might carry some unique specifications that could result in advantages if combined with other individuals. Roulette wheel selection is a commonly used selection operator which calculates the probability of selecting each individual based on its fitness value. For a solution x in population P, the likelihood of being chosen as a parent is $f(x)/(\sum_{y\in P} f(y))$. The other selection operator in many standard EAs is tournament selection. Tournament selection picks two individuals uniformly at random and selects the better one as the parent.

The number of parents to be chosen depends on the reproduction operators, which determine how to combine parents to create offspring. Usually, two main operators (*crossover* and *mutation*) are used to perform this step. These operators are described in the next sections.

2.3.3 Crossover

The crossover is responsible for recombination of the parents, and it results in *inheritance*. In other words, crossover operation transfers the properties of the parents to the children. The standard crossovers take two individuals as parents and create two offspring. For bit-string representations, the single-point crossover and uniform crossover are commonly used operators. Consider two solutions $x = (x_1, \cdots, x_n)$ and $y = (y_1, \cdots, y_n)$. The single-point crossover operator chooses a random number $1 \leq i \leq n-1$, cuts both solutions from the ith bit, and swaps the sub-parts. The result of this operation for x and y will be $xy^1 = (x_1, \cdots, x_i, y_{i+1}, \cdots, y_n)$ and $xy^2 = (y_1, \cdots, y_i, x_{i+1}, \cdots, x_n)$. The uniform crossover, on the other hand, selects the ith bits of xy^1 and xy^2 from $\{x_i, y_i\}$, uniformly at random. This operator randomly chooses a bit for the first offspring and sets the other bit for the second offspring. For the LEADINGONES problem, the following example shows how a low-quality solution might carry beneficial properties. The crossover operation is the key to transfer those properties to the next generation. Imagine there exists a solution that has a series of zero bits at the beginning, followed by a long sequence of ones. While the fitness value of this solution is minimal, using a single-point crossover to combine it with an appropriate solution can result in a high-quality individual. Note that this example also shows why the selection operator must consider some small probability for selecting also bad solutions.

2.3.4 Mutation

The next step is to perform mutation. In contrast to the crossover (which transfers the specifications), the mutation operator ensures that the population is not trapped in a local area. In many situations, because of the dominance of good solutions and their high chance of being selected for reproduction, most of the individuals converge on the best-found solution. However, we need the population to cover all of the search space, in order to find the global optima rather than converging on local

ones. The mutation operator makes an entirely random change on an offspring. This means it might develop a solution that is far from the original one, either in the solution space or in objective space. To perform the mutation, evolutionary algorithms need a (small) mutation probability that is set as a parameter of the algorithm. After the crossover produces offspring, the algorithm considers each of the offspring and mutates them according to the mutation probability. For bit-string solutions with size n, a typical mutation operation is to flip each bit of the solution with the probability of $1/n$. One can also use the search operator in RLS, which selects one bit of the solution uniformly at random and flips it. For solutions with real values, a mutation operator can add or subtract a random value to or from the original solution.

2.3.5 Survivor Selection

Finalising the offspring reproduction, EAs must select final solutions to create a new population. Depending on the number of generated offspring and the size of the population, there are different strategies to perform this step. In contrast to the parent selection discussed in Section 2.3.2, all these strategies are deterministic. In this step, the algorithms either compare offspring with the parents and select the ones with better fitness value, or only select the next generation from the offspring solutions. Moreover, some algorithms apply elitism, transferring the best of the previous population directly to the next population. This could also include transferring some random solutions to the next population, in order to ensure random inheritance.

The evolutionary loop continues until some threshold is met. The threshold entails one of the following: either the algorithm reaches a pre-set number of generations, the algorithm performs a determined number of evaluations, or it fails to make any significant additional improvements to the quality of the results.

2.3.6 (1+1) EA

The performance of different evolutionary approaches may vary according to the nature of the problem and the specifications of the search space. In this thesis, we study the performance of $(1+1)$ EA (see Algorithm 3) as a simple single-objective evolutionary algorithm in which the population includes only one solution and only one offspring is generated at each iteration. This algorithm is quite similar to RLS, except that multiple bit flips are allowed in the mutation step. Instead of flipping one bit uniformly at random, this algorithm has the potential to flip all bits of the current solution with probability $1/n$, where n is size of the solution.

Note that $(1+1)$ EA does not use a crossover, and it explores the search space by implementing the mutation operator only. However, the fact that it might flip more than one bit significantly improves its performance in comparison to RLS. Moreover, we show in Chapter 7 that in optimising only one solution, the $(1+1)$ EA is superior to some advanced multi-objective optimisers in specific dynamic environments.

Algorithm 3: $(1+1)$ EA

```
The initial solution x is given;
while stopping criterion not met do
    y ← flip each bit of x independently with probability 1/n;
    if f(y) ≥ f(x) then
        x ← y;
```

2.3.7 Multi-Objective Evolutionary Algorithms

In many real-world applications, we face problems with more than one objective function to be optimised. Such problems are termed *multi-objective* problems. Let k denote the number of objectives. Then each solution x will be a k-dimensional point $f(x) = (f_1(x), \cdots, f_k(x))$ in the objective space. In such cases, comparison of the solutions might not be as clear as is the case for single-objective problems. This is because one solution could be better in one objective but worse in another.

A simple approach to dealing with more than one objective is to consider weights for each objective and solve the single-objective version. In a maximisation problem, let w_i denote a real weight value for objective function f_i. One can use a single-objective optimisation algorithm to maximise $g(x) = \sum_{i=1}^{n} w_i f_i(x)$. This approach, called *weighted-sum*, is highly dependent on the weight vector $(w_1, \cdots, w_k)$. Since the weighted-sum approach finds an optimal solution in the direction of the weight vector, it cannot produce good results to complex problems with non-convex objective spaces. Moreover, in multi-objective problems, we are looking for a handful of solutions that are optimal in terms of different objectives. Hence, we need to use the weighted-sum approach multiple times, with different weight vectors. However, there is no guarantee that well-distributed weight vectors result in a good distribution of the solutions in objective space [FF95].

A more promising way to address the quality problem in multi-objective optimisation is the *dominance* concept. For a maximisation problem, we say that solution x dominates y ($x \succeq y$) if and only if $x \neq y$ and $f_i(x) \geq f_i(y)$ for $1 \leq i \leq k$. The dominance is strong and denoted by $x \succ y$ if at least one inequality is strict.

Note that according to the definition of dominance, there is not necessarily any single optimal solution. This is because the objectives might contradict each other. For more clarification, consider a multi-objective version of LEADINGONES called LEADINGONES-TRAILINGZEROS [Lau+02]. In this version we want to optimise the objective function $LOTZ(x) = (LO(x), TZ(x))$. In addition to $LO(x) = \sum_{j=1}^{n} \prod_{i=1}^{j} x_i$, the second objective $TZ(x) = \sum_{j=1}^{n} \prod_{i=j}^{n} (1 - x_i)$ is defined as the length of a sequence of zero bits that is started from the last position, where the aim is to maximise both objectives. Clearly, maximising $LO(x)$ in this problem, minimises $TZ(x)$.

In multi-objective problems, the goal is to find a set of optimal solutions. These

Algorithm 4: G-SEMO Algorithm

```
1 The initial solution x is given;
2 P ← {x};
3 while stopping criterion not met do
4     Select x from P uniformly at random;
5     x' ← flip each bit of x with probability 1/n;
6     if ∄z ∈ P such that z ≻ x' then
7         P ← (P \ {z ∈ P | x' ⪰ z}) ∪ {x'};
```

solutions, called *non-dominated* solutions or *Pareto optimal* solutions, are solutions which are not dominated by any other solution in the search space. The Pareto solutions to the LEADINGONES-TRAILINGZEROS problem, for example, are a set of $n+1$ solutions $\{s_0, \cdots, s_n\}$, where s_i starts with a sequence of i one bits followed by a sequence of $n-i$ zero bits. Note that the size of the *Pareto set* could be exponential for some discrete optimisation problems. However, the algorithms try to find a well-distributed set that includes a solution for all parts the *Pareto front*, the map of a Pareto set in the objective space.

Attempts to optimise multi-objective problems while considering all the objectives separately (and without converting them into a scalar optimisation) began with the vector evaluated genetic algorithm (VEGA) in 1984 [Sch85]. The most significant part of VEGA was the development of a selection operator that randomly divides the population into k subsets, where k is the number of objective functions. For the kth subset, the comparison of solutions to be chosen for the reproduction phase was based on the kth objective. In this algorithm, a solution with a considerable fitness value in the jth function had to be lucky to be compared based on the jth function. Otherwise, it could not survive. Later, Murata and Ishibuchi introduced the multi-objective genetic algorithm (MOGA), which combines the weighted-sum approach with the dominance concept [MI95]. For the current population, MOGA finds the non-dominated set first. In the selection phase, the algorithm produces a random weight vector, re-evaluates the population based on the $g(x)$ function and then assigns each solution a proportional selection probability according to the new fitness values. After performing the crossover and mutation operators and generating the new population, MOGA replaces a fixed number of randomly chosen individuals from the randomly chosen solutions of the non-dominated set. This approach increases the survival chance of Pareto solutions.

In the next two sections, we describe the details of the multi-objective approaches that we consider in this thesis, both theoretically and empirically.

Algorithm 5: NSGA-II

```
1  Generate initial population set P_0 and offspring set Q_0;
2  while stopping criterion not met do
3      R_t ← P_t ∪ Q_t;                    // combine parent and offspring population
4      F ← fast-non-dominated-sort(R_t);    // F = (F_1, F_2, ···), all non-dominated fronts of R_t
5      P_{t+1} ← ∅ and i ← 1;
6      while |P_{t+1}| + |F_i| ≤ N do
7          crowding-distance-assignment(F_i);
8          P_{t+1} ← P_{t+1} ∪ F_i;
9          i ← i + 1;
10     Sort F_i based on the crowding distance in descending order;
11     P_{t+1} ← P_{t+1} ∪ F_i[1 : (N − |P_{t+1}|)];
12     Q_{t+1} ← make-new-pop(P_{t+1});
13     t ← t + 1;
```

G-SEMO

The global simple evolutionary multi-objective algorithm (G-SEMO) introduced in [Gie03] is a simple extension of the $(1+1)$ EA algorithm for multi-objective optimisation (see Algorithm 4). Its population is initialised with only one solution. In the main loop, G-SEMO selects a random solution x from the population and mutates it by using the same mutation operator in $(1+1)$ EA. The created solution y is then compared to other individuals based on the dominance concept introduced earlier in this section. G-SEMO looks for solutions that are not strongly dominated by other solutions in the population. If the algorithm finds a non-dominated solution, it adds it to the population and removes the solutions that are (weakly) dominated by the new one. In other words, this algorithm only keeps non-dominated solutions; this is how it controls its population size. However, the size may become exponential. In cases of potentially exponential population growth, there are some approaches that categorise solutions based on their property (their size for instance), as well as one used in Chapter 6 to deal with potentially exponential population size. As discussed earlier, another challenge in multi-objective problems is to find non-dominated solutions that represent all the Pareto front. G-SEMO, however, does not use any particular approach to maintain the diversity of the Pareto set. In the next section, we introduce two algorithms that use specific approaches for generating well-distributed solutions.

NSGA-II and SPEA2

The first version of the non-dominated sorting genetic algorithm (NSGA) was introduced in 1995 [SD94]. The main innovation in this algorithm was to consider the population set as a set of different Pareto fronts. The solutions were then selected based on the front in which they had been. However, NSGA's survival loop,

Algorithm 6: SPEA2

Generate the initial population set P_0 and archive set $\overline{P_0}$={};
Calculate the fitness values of solutions in P_0;
while *stopping criterion not met* **do**
    Mating selection: Generate a mating pool by tournament selection from $\overline{P_t}$;
    Variation: Apply crossover and mutation operators on the mating pool to produce P_{t+1};
    Fitness assignment: Calculate fitness values of solutions in P_{t+1} and $\overline{P_{t+1}}$;
    Environmental selection: Generate $\overline{P_{t+1}}$ by choosing $\overline{N}$ non-dominated solutions from P_t and $\overline{P_{t+1}}$;

similar to the MOGA, had a high chance of loosing valuable solutions. Later in 2002, NSGA-II was introduced [Deb+02]. In the initial state, NSGA-II starts with a randomly generated population P_0 and assigns a fitness (or rank) to each solution based on its non-dominated rank. Offspring population Q_0 is then produced using usual selection, recombination, and mutation operators. Algorithm 5 describes how NSGA-II performs after the initial step and after each dynamic change. It sorts the combination of offspring and population sets into a set of non-dominated fronts $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \cdots\}$ such that solutions in $\mathcal{F}_1$ are non-dominated solutions, solutions in $\mathcal{F}_2$ are non-dominated solutions after removing solutions of $\mathcal{F}_1$ from the combined set, and so on. Then, starting from the first front, it adds solutions to P_{t+1} until the ith front which adding $\mathcal{F}_i$ exceeds the population size N. NSGA-II calculates crowding distance for each of the solutions, in order to distinguish solutions in the same fronts. The crowding distance is an estimation of the perimeter of a cuboid around a solution formed by using the solution's nearest neighbours as vertices. Thus, larger crowding distance means that the solution is in a sparse area. It assigns an infinite value to the solutions with the highest or lowest objective values in each front. The algorithm finally produces Q_{t+1} from P_{t+1}, using evolutionary operators and considering the front rank and crowding distance as the objectives.

SPEA2, an improved version of the strength Pareto evolutionary algorithm introduced in [ZT98], uses another approach to produce distributed solutions [ZLT01]. As described in (Algorithm 6), this algorithm keeps the best non-dominated solutions of each generation in the archive set $\overline{P_t}$ with size $\overline{N}$, and generates population set P_{t+1} by performing evolutionary operators on $\overline{P_{t+1}}$. The fitness value of solution x in SPEA2 is calculated based on two factors: integer raw fitness $0 \leq R(x)$, which represents the non-dominancy power of solutions that dominate x, and density estimate $0 < D(x) \leq 1/2$, which is calculated based on the inverse of distance to the kth nearest neighbour of x with the same raw fitness value. $k = \sqrt{N + \overline{N}}$ is chosen as the default value of k. The final fitness value of x is $F(x) = R(x) + D(x)$, in which $R(x)$ is 0 when x is a non-dominated solution and the lesser value of $D(x)$ illustrates the better distributed solution.

2.4 Analytical Methods

The history of runtime analysis of bio-inspired computing begins with Muhlenbein's study of evolutionary algorithms [Müh92]. The computational analysis of these algorithms considers the number of evaluations that is required to find an optimal solution with respect to the size of the input. There have been many advancements in the theoretical tools which can be used to perform rigorous runtime analysis of bio-inspired computing. These tools have been used to analyse the behaviour of evolutionary algorithms and ant colony optimisation solving combinatorial optimisation problems [Doe20; Len20]. The following sections introduce techniques and analytical tools that we used in this thesis to study the performance of algorithms.

2.4.1 Deviation Bounds

Deviation inequalities are the well-known mathematical tools that are mainly used to bound the probability of deviating a random variable from its expected value. In the area of runtime analysis, these tools are used to prove the probability that the actual running time of the optimisation algorithms will deviate from the expected time. In this section, we introduce the two commonly used deviation inequalities.

Markov's Inequality: For a non-negative random variable X and for all $k \in \mathbb{R}_{>0}$, the following inequality holds

$$\Pr(X \geq k \cdot E[X]) \leq \frac{1}{k}$$

Note that X could be any non-negative random variable. Thus, Markov's inequality is specifically useful when we can repeat a process to ensure the result.

Chernoff Bounds Let $X_1, X_2, \cdots, X_n$ be independent Poisson trials such that, for $1 \leq i \leq n$, $\Pr(X_i = 1) = p_i$, where $0 \leq p_i \leq 1$, and $\mu = E[X] = \sum_{i=1}^{n} p_i$. Then, the following inequalities hold:

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

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

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

Chernoff bounds are exponentially decreasing bounds which are stronger than the Markov's inequality. However, they require the random variables to be independent.

2.4.2 Fitness Based Partitions

This simple method introduced by Wegener [Weg03] in 2003 partitions the search space into smaller parts with some specific ranked properties, and calculates the expected time of passing through these partitions to find the optimal solution. To be more precise, let D denote the search space and $f : D \to R$ be the objective function to be maximised. Dividing D into m partitions, denoted by $A_1, A_2, \cdots, A_m$, assume that for any $i < j$, $x \in A_i$, and $y \in A_j$ we have $f(x) < f(y)$. In other words, the fitness value of solutions increases when increasing the index of partitions. Furthermore, let A_m only include optimal solutions. Let $p(x)$ be the probability of creating solution $y \in A_j$ from solution $x \in A_i$ such that $j > i$. Considering $p_i = \min_{x \in A_i} p(x)$, the smallest probability that jumping from A_i to A_j will happen, Lemma 1 in [Weg03] proves that the expected optimisation time is upper bounded by $\sum_{i=1}^{m-1}(1/p_i)$. The associated proof is simple. Since the smallest probability of producing a solution in partitions $A_{i+1}, \cdots, A_m$ from a solution of partition A_i is p_i, the expected time required to arrive at such an event $1/p_i$.

We now show the application of this method using a simple example. Consider the LEADINGONES problem introduced in Section 2.1 and the RLS algorithm in Section 2.2.1. The search space of the LEADINGONES problem is $\{0,1\}^n$, the set of all bit-strings with size n. Let A_i represent the set of bit-strings with fitness i, i. e., solutions that start with exactly i but not $i+1$ consecutive ones. To improve the fitness of solution $x \in A_i$, RLS must flip the $(i+1)$th bit of x. This occurs with probability $1/n$. Thus, the expected time in which RLS will find the optimal solution to the LEADINGONES problem is upper bounded by $\sum_{i=1}^{n-1} n = O(n^2)$.

2.4.3 Drift Analysis

Drift analysis (introduced by He and Yao [HY01]) is one of the most important tools in theoretical analysis of bio-inspired computing. Generally speaking, this method counts the number of steps required for the algorithm to fill a gap defined by an auxiliary function.

The simplified formal definition of the drift is as follows:

Definition 2.1 (Drift [LW13])**.** *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 $\delta = E[X_t - X_{t+1} \mid F_t]$ for $t > 0$ is called drift.*

After defining the auxiliary function, the goal is to show that the algorithm can improve the value of the function by at least a certain amount in each step. When the amount of improvement is constant, one can use the additive drift, which is defined as follows:

Definition 2.2 (Additive Drift [LW13]). *Let $(X_t)_{t\geq 0}$, be a stochastic process over some bounded state space $S \in \mathbb{R}_0^+$. Assume that $E[T_0 \mid X_0] < \infty$, where $T_a = \min\{t \mid X_t \leq a\}$ is the first hitting time for threshold $a \geq 0$. Then:*

- *if $E[X_t - X_{t+1} \mid F_t; X_t > 0] \geq \delta_u$ then $E[T_0 \mid X_0] \leq \frac{X_0}{\delta_u}$.*
- *if $E[X_t - X_{t+1} \mid F_t] \leq \delta_l$ then $E[T_0 \mid X_0] \geq \frac{X_0}{\delta_l}$.*

Using additive drift, we now analyse the performance of RLS on LEADINGONES. We set the random variable $X_t = n - LO_1(x^t)$, where x^t is the current solution of RLS in step t. Since RLS flips exactly one bit of x^t, the probability of flipping a zero bit in position $LO_1(x^t) + 1$ is at least $1/n$. Thus, we have $E[X_t - X_{t+1}|X_t] \geq \delta_u = 1/n$. In the worst case, the initial solution of RLS does not have any one bits, i. e., we have $X_0 = n$. Hence, using additive drift, the expected time for RLS to find the optimal solution to the LEADINGONES problem is $E[T_0|X_0] \leq X_0/\delta_u \leq n^2 = O(n^2)$.

The more recently introduced multiplicative drift (which we also use in this thesis) considers the improvements that are proportional to the current value of the auxiliary function. The definition of multiplicative drift as introduced in [DJW12] is as follows:

Theorem 2.3 (Multiplicative Drift [DJW12]). *Let $S \in \mathbb{R}$ 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 $\Pr(X^{(t)} = s) > 0$. Then for all $s_0 \in S$ with $\Pr(X^{(0)} = s_0) > 0$, we have

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

To demonstrate how to use multiplicative drift, we briefly define the ONEMAX problem (see Section 3.2.1 for a detailed definition). For a given bit-string $x \in \{0,1\}^n$, $OM(x) = \sum_{i=1}^{n} x_i$ is the number of ones in x. The goal is to find a solution $x^* = (1, \cdots, 1)$ that maximises OM.

Now, we use multiplicative drift to prove an upper bound on the expected time required for RLS to optimise ONEMAX. For a given solution x^t, let $s = n - OM(x)$ denote the number of zeros in x^t. Considering the same definition of $X_t = s$, the probability of reducing s by one is equal to the probability of flipping a zero bit, which is s/n. Hence, we have $E[X_t - X_{t+1}|X_t = s] = s/n$. Note that the the value of s_0 is n in the worst case and the smallest value of s is $s_{\min} = 1$. Setting δ to $1/n$, we can use multiplicative drift to show that the expected time for RLS to find the optimal solution of ONEMAX is $E[T|X_0 = n] = (1 + \ln(n))\,/(1/n) = O(n \log n)$.

2.4.4 Conclusion

The first part of this chapter presented details of the bio-inspired algorithms that we use in this thesis. It began by introducing the local search method RLS, and then described the main concepts of evolutionary computation. Next, the simplest evolutionary algorithm and its extension for multi-objective optimisation, known as $(1+1)$ EA and G-SEMO respectively, were presented. At the end of the first part, we introduced NSGA-II and SPEA2, which are well-known advanced multi-objective algorithms.

In the second part of this chapter, we reviewed analytical methods for studying bio-inspired computation. We introduced fitness-based partitioning, which tracks the quality of solutions in moving through defined levels after presenting deviation bounds that are used to bound the probability of deviating from the expected value. Finally, we presented more recently introduced analytical tools (known as drift) that consider the amount of improvement in the quality of solutions in each step of the algorithms.

Chapter 3

Combinatorial Optimisation

3.1 Introduction

Combinatorial optimisation aims to find the best feasible solution in a finite discrete search space when a fitness function evaluates solutions and a set of given constraints determines the search space [PS82]. Combinatorial optimisation problems appear in many applications, such as facility location, scheduling and AI. As a general definition, a combinatorial optimisation problem is characterised by a triple (S, f, Ω), where S denotes the search space, f denotes the fitness function and Ω is the set of constraints. The goal is to find the global optimal solution in the search space such that the solution optimises (maximises or minimises) the fitness function and satisfies the constraints.

The size of the search space in many combinatorial optimisation problems grow exponentially in relation to the size of input. While in some cases an algorithm exists that finds the optimal solution(s) in polynomial time, there is no such algorithm for most of the combinatorial optimisation problems unless $\mathcal{P} = \mathcal{NP}$. For these $\mathcal{NP}$-hard problems, we are looking for the algorithms that guarantee an approximation of the optimal solutions. The precise definition of an α-approximation ratio is as follows.

Definition 3.1 (Approximation ratio)**.** *An algorithm for a problem has an approximation ratio of $\alpha(n)$ if, for any input of size n, the fitness OPT of the solution produced by the algorithm is within a factor of $\alpha(n)$ of the fitness OPT^* of an optimal solution:*

$$\max\left(\frac{OPT}{OPT^*}, \frac{OPT^*}{OPT}\right) \leq \alpha(n).$$

The knapsack problem, minimum vertex cover problem and subset selection problem are all examples of problems that are considered in this thesis and have an approximation algorithm with a constant approximation ratio. We also study the

packing while travelling problem, which is proven not to have any approximation algorithm with a constant approximation ratio [PN15].

In this chapter, we define the problems that are considered in this thesis, including the problems studied in Chapters 5 to 8 and the problems that are reviewed in the survey Chapter 4. Firstly, we define a broad set of optimisation problems known as linear pseudo-Boolean functions, as well as the ONEMAX problem, which is an important example of this type. Sections 3.3 and 3.4 introduce the knapsack problem and packing while travelling problem, both of which are packing problems with different complexities. Next, we present the minimum spanning tree and minimum vertex cover problems in Sections 3.5 and 3.6, respectively. These are two well-known graph problems. Finally, after presenting the makespan scheduling problem in Section 3.7, we define a problem called 'subset selection' in Section 3.8. The definition of subset selection includes all the other problems introduced in this chapter.

3.2 Linear Pseudo-Boolean Functions

Linear pseudo-Boolean functions play a key role in the runtime analysis of evolutionary algorithms. Let $x = (x_1, x_2, \ldots, x_n)$ be a search point in search space $\{0,1\}^n$, and w_i, $1 \leq i \leq n$ positive real weights. A linear pseudo-Boolean function $f(x)$ is defined as follows:

$$f(x) = w_0 + \sum_{i=1}^{n} w_i x_i.$$

For simplicity and as done in most studies, we assume $w_0 = 0$ in the rest of this section. The optimisation of a linear objective function under a linear constraint is equivalent to the classical knapsack problem [KPP04] (see Section 3.3). The optimisation of a linear objective function together with a uniform constraint has recently been investigated in the static setting [Fri+17]. Given a bound B, $0 \leq B \leq n$, a solution x is feasible if the number of 1-bits of the search point x has a maximum of B. The bound B is also known as the *cardinality bound*. We denote the number of 1-bits of x by $|x|_1 = \sum_{i=1}^{n} x_i$. The formal definition for maximising a pseudo-Boolean linear function under a cardinality bound constraint is given by:

$$\begin{aligned} &\max f(x) \\ &\text{s.t. } |x|_1 \leq B. \end{aligned}$$

The dynamic version of this problem, referred to as the problem with a *dynamic uniform constraint*, is defined in [Shi+19]. Here, the cardinality bound changes from B to some new value B^*. Starting from a solution that is optimal for the bound B, the problem is then to find an optimal solution for B^*. The re-optimisation time of an evolutionary algorithm is defined as the number of fitness evaluations that are

required to find the new optimal solution. In Section 4.2.2, we present the most recent results in optimising linear pseudo-Boolean functions under dynamic uniform constraint.

3.2.1 ONEMAX Problem

As a specific case of a linear pseudo-Boolean function with weights equal to one, ONEMAX has gained a lot of attention in the area of runtime analysis. In the ONEMAX problem, the number of ones in the solution is the objective to be maximised. More precisely, for a bit-string solution $(x_1, \cdots, x_n) \in \{0,1\}^n$, the goal is to maximise the objective function $f(x) = \sum_{i=1}^{n} x_i$. Droste [Dro02] has interpreted this problem as maximising the number of bits that match a given objective bit-string. With this goal in mind, he has introduced the dynamic ONEMAX problem, in which dynamic changes happen to the objective bit-string over time. An extended version of this problem is defined by Kötzing et al. [KLW15]. In their version of the problem, not only are bit-strings allowed, but each position can take on integer values in $\{0, \ldots, r-1\}$ for $r \in \mathbb{N}_{\geq 2}$. The formal definition of the problem follows.

Let $[r] = \{0, \ldots, r-1\}$ for $r \in \mathbb{N}_{\geq 2}$, and $x, y \in [r]$. Moreover, let the distance between x and y be

$$d(x,y) = \min\{(x-y) \mod r, (y-x) \mod r\}.$$

The extended ONEMAX problem, $\text{ONEMAX}_a : [r]^n \to \mathbb{R}$, where a is the objective string defining the optimum, is given as:

$$\text{ONEMAX}_a(x) = \sum_{i=1}^{n} d(a_i, x_i).$$

The goal is to find and maintain a solution with minimum value of ONEMAX_a.

Given a probability value p, the dynamism that is defined on this problem is to change each component i, $1 \leq i \leq n$, of the optimal solution a independently as:

$$a_i = \begin{cases} a_i + 1 \mod r; & \text{with probability } p/2 \\ a_i - 1 \mod r; & \text{with probability } p/2 \\ a_i; & \text{with probability } 1-p \end{cases}$$

The detailed results of analysing the dynamic ONEMAX problem have been presented in Section 4.2.1

3.3 Knapsack Problem

This section introduces the knapsack problem (KP), which has been studied extensively in the past decades because of its many applications [RT89; Sah75]. In this problem, n items with profits $\{p_1, \ldots, p_n\}$ and weights $\{w_1, \ldots, w_n\}$, and a knapsack

with capacity C are given. The goal is to fill the knapsack with the items in order to maximise the total profit while ensuring that the total weight does not exceed C.

The fractional KP is an easy version of this problem which is solvable in polynomial time with a deterministic algorithm. In this version, it is permissible to break the items. Thus, it is sufficient to select items in a 'greedy' manner, based on the profit/weight ratio. However, this greedy algorithm can perform arbitrarily badly in the 0/1 KP. In the 0/1 KP, we can either pick an item or leave it. This version is proven to be $\mathcal{NP}$-hard (a simple modification in the greedy algorithm results in a 2-approximation algorithm for 0/1 KP). In this version, a solution x could be represented by a bit string of $\{0,1\}^n$ which has the overall weight $w(x) = \sum_{i=1}^{n} w_i x_i$ and profit $p(x) = \sum_{i=1}^{n} p_i x_i$. The goal is to find a solution $x^* = \arg\max\{p(x) \mid x \in \{0,1\}^n \wedge w(x) \leq C\}$ of maximum profit of which the weight does not exceed the capacity constraint C. In the rest of this thesis, we only consider the 0/1 knapsack problem, referring to it as the 'knapsack problem'.

In the following section, we introduce the dynamic programming approach to solve KP in pseudo-polynomial time $O(n^2 P_{\max})$ where $P_{\max} = \max\{p_1, \cdots, p_n\}$. Consider table A with size $n \times nP_{\max}$. Let $A[i,j]$ be the weight of a solution that is a subset of first i items with profit j and the minimum weight. Then we set

$$A[i+1,j] = \begin{cases} \min\{A[i,j], w_{i+1} + A[i,j-p_{i+1}]\} & \text{if } p_{i+1} \leq j \\ A[i,j] & \text{otherwise.} \end{cases}$$

Finally, the optimal solution will be $\max\{j | A[n,j] \leq C\}$. This is a pseudo-polynomial algorithm, since $P_{\max}$ could be exponential in size n, e. g., 2^n. However, this approach has been used to propose a fully polynomial time approximation scheme (FPTAS) for KP. With this aim, instead of the actual profits p_i, we use $p' = \lfloor p_i/K \rfloor$, where $K = \varepsilon P_{\max}/n$ for an arbitrary small epsilon. It is shown that this algorithm results in a $(1-\varepsilon)$-approximate solution in time $O(n^2 \lfloor n/\varepsilon \rfloor)$.

The dynamic version of KP is investigated empirically in Chapter 7. We study dynamic environments in which the capacity of the knapsack changes during the optimisation process. Different scenarios are considered according to the magnitude, frequency and distribution of dynamic changes.

3.4 Packing While Travelling

In many real-world applications, we are facing problems that consist of multiple sub-problems (or components). The relation between the components is such that to solve the main multi-component problem, it is necessary to consider all the components simultaneously.

Examples of multi-component problems include the generalised minimum spanning tree problem (GMST) and the generalised travelling salesman problem (GTSP).

In both of these cases, clusters of nodes exist; the problem is to choose one node from each cluster, and then solve the MST and TSP on the selected nodes, respectively. These problems can be decomposed into a node selection component and an MST/TSP component. Hierarchical approaches to each problem are investigated theoretically in [Cor+16] and [PN18].

The travelling thief problem (TTP), introduced by Bonyadi et al., is another example of an $\mathcal{NP}$-hard multi-component problem that combines the travelling salesman problem and the knapsack problem [BMB13]. In this problem, there are a number of items distributed between the nodes of a graph. The goal is to find a Hamiltonian cycle and choose items from the visited nodes such that the benefit function is maximised. The integration of these components in a non-linear objective function is carried out in such a way that optimising one component does not necessarily result in a near-optimal solution.

Using the same formulation, Polyakovskiy and Neumann introduced the packing while travelling problem (PWT), which is actually the packing version of TTP with a fixed travelling path [PN17]. The goal in this problem is to a packing plan for a vehicle with a specified capacity. The vehicle must visit all the nodes in a specific order, and pick items in a way that maximises the benefit function without violating the capacity constraint. The benefit function has a direct relationship to the profit of packed items, while its relationship with velocity is inverse. The velocity of the vehicle as it travels between each of the two cities is determined by the weight of the vehicle, such that the more weight the vehicle carries, the lower its velocity. Polyakovskiy and Neumann proved that this problem, for even two cities, is $\mathcal{NP}$-hard. Later, Neumann et al. [Neu+18] presented an exact dynamic programming approach for PWT. They also proved that there is no polynomial time algorithm with constant approximation ratio for PWT unless $\mathcal{P} = \mathcal{NP}$, and presented an FPTAS for maximising the objective value over the baseline travel cost in which the vehicle travels the path empty.

I the general PWT problem proposed in 2016 by Polyakovskiy and Neumann [PN15], given a set of $m+1$ ordered cities, distances d_i from city i to $i+1$ ($1 \leq i \leq m$) and a set of items $N = \cup_{i=1}^{m} N_i$ distributed over first m cities such that city $1 \leq i \leq m$ contains items N_i, let $|N_i| = n_i$ denote the number of items in city i. Positive integers profit p_{ij} and weight w_{ij} are assigned to each item $e_{ij} \in N_i$, $1 \leq j \leq n_i$. Path $M = (1, 2, \ldots, m+1)$ is travelled by a vehicle with velocity $v = [v_{\min}, v_{\max}]$ and capacity C. A solution vector $s = (x_{11}x_{12}\ldots x_{1n_1}\ldots x_{mn_m})$ represents a set of selected items $S \subseteq N$ such that variable $x_{ij} \in \{0, 1\}$ indicates whether item e_{ij} is selected or not. Let $W(s)$ denote the sum of weights of items in s. As such, s is feasible if $W(s) \leq C$. Let

$$P(s) = \sum_{i=1}^{m} \sum_{j=1}^{n_i} p_{ij} x_{ij}$$

be the total profit and

$$T(s) = \sum_{i=1}^{m} \frac{d_i}{v_{\max} - \nu \sum_{k=1}^{i} \sum_{j=1}^{n_k} w_{kj} x_{kj}},$$

where $\nu = \frac{v_{\max} - v_{\min}}{C}$ is a constant, be the total travel time for a vehicle carrying items defined by s. The denominator of $T(s)$ is such that picking an item in city i only affects the time required to travel from city i to the end. Thus, the benefit value of s is computed as

$$B(s) = P(s) - R \cdot T(s),$$

where R is a given renting rate. The aim is to find a feasible solution s^* such that $s^* = \arg\max_{s \in \{0,1\}^n} B(s)$.

The effect that different values of R have on the benefit function has been considered previously [WPN16]. Generally speaking, where $R = 0$, PWT becomes a $0/1$ knapsack problem, while a larger R entails a situation in which the vehicle must pick fewer items to arrive at the optimal solution. This problem is proven to be $\mathcal{NP}$-hard by reducing the subset sum problem to the decision variant of unconstrained PWT [PN17].

Chapter 6 theoretically investigates the performance of variants of RLS, $(1+1)$ EA, and G-SEMO algorithms on a version of PWT involving two cities.

3.5 Minimum Spanning Tree Problem

The minimum spanning tree (MST) is a subset of the edges of an undirected connected graph G with specific features. A spanning tree of graph G is a sub-graph G' if and only if there exists exactly one path between any two connected vertices in G'. Each edge e is assigned a positive weight $w(e)$ and the goal is to find a spanning tree with minimum total weight, called MST. Note that MST is not necessarily unique. However, it is widely used in real-world applications such as path planning and clustering. To be more precise, for a given graph $G = (V, E)$ with vector set V and edge set E, and a given weight function $w : E \to \mathbb{R}^+$, the minimum spanning tree of G is spanning tree $T = (V, E')$ such that $\sum_{e \in E'} w(e)$ is minimum.

The MST problem is well-understood and solvable in polynomial time using well-known algorithms, e.g., Kruskal's algorithm [Kru56]. His algorithm starts with an empty graph G'' that doesn't have any edges, but has the same vertices as the given graph. Then it sorts the edges of E in ascending order, based on their weights. At each step, the algorithm adds the lightest edge to G'' if it does not cause a cycle. The process continues until G'' becomes a spanning tree, which is proven to also be an MST.

In the area of runtime analysis of bio-inspired computation, spanning tree problems have obtained significant attention. The classical MST problem has been investigated for simple single-objective approaches of EAs [NW07] and ant colony optimization [NW10a]. Furthermore, it has been shown that a multi-objective formulation of the problem can lead to significantly faster evolutionary algorithms [NW06].

In Chapter 5, we investigate the performance of evolutionary algorithms enhanced with a knowledge-based biased mutation which faces single- and multi-objective MST problem.

3.6 Vertex Cover Problem

The vertex cover problem is one of the best-known $\mathcal{NP}$-hard combinatorial optimisation problems. Given a graph $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$ is the set of vertices and $E = \{e_1, \ldots, e_m\}$ is the set of edges, the goal is to find a minimum subset of nodes $V_C \subseteq V$ that covers all edges in E, i.e., $\forall e \in E, e \cap V_C \neq \emptyset$. In the dynamic version of the problem, an edge can be added to or deleted from the graph.

As the vertex cover problem is $\mathcal{NP}$-hard, it has been mainly studied in terms of approximations. The problem can be approximated within a worst-case approximation ratio of 2 by various algorithms. One standard approach to obtaining a 2-approximation is to compute a maximal matching and take all nodes adjacent to the chosen matching edges for the vertex cover. In the dynamic version of the problem (starting from a solution that is a 2-approximation for the current instance of the problem) the goal is to obtain a 2-approximate solution for that instance of the problem after one dynamic change. The re-optimisation time for this problem refers to the time required for the investigated algorithm to find a 2-approximate solution for the new instance. This dynamic setting has been investigated in [PGN15] and the detailed explanation of the results is presented in Section 4.2.3.

3.7 Makespan Scheduling Problem

The other classical problem that we consider in our survey (see Chapter 4) is the makespan scheduling problem [SN12], which can be defined as follows. Given n jobs and their processing times $p_i > 0, 1 \leq i \leq n$, the goal is to assign each job to one of two machines M_1 and M_2 such that the makespan is minimised. The makespan is the time that the busier machine takes to finish all assigned jobs. A solution is represented by a vector $x \in \{0,1\}^n$ which means that job i is assigned to machine M_1 if $x_i = 0$ and it is assigned to M_2 if $x_i = 1, 1 \leq i \leq n$. With this representation, the makespan of a given solution x is given by

$$f(x) = \max \left\{ \sum_{i=1}^{n} p_i(1 - x_i), \sum_{i=1}^{n} p_i x_i \right\}$$

and the goal is to minimise f. In the dynamic version of this problem, the processing time of a job may change over time, but stays within a given interval. Neumann and Witt [NW15] have investigated the setting $p_i \in [L, U]$, $1 \leq i \leq n$, where L and U are a lower and upper bound for each processing time. Their analysis concentrates on the time evolutionary algorithms need to produce a solution where the two machines have a time discrepancy at most U. Dynamic changes to the processing times of the jobs have been investigated in two different settings. In the first setting, an adversary is allowed to change the processing time of exactly one job. In the second setting, the job to be changed is picked by an adversary but the processing time of a job is altered randomly.

3.8 Subset Selection and Submodular Functions

Most of the problems that we have introduced so far are specific examples of a more general $\mathcal{NP}$-hard problem called the subset selection problem. Given a set of elements, the goal is to select a subset of these elements to maximise an objective function according to a cost function and a budget constraint. The subset selection problem appears in many real-world application with a variety the objective and cost functions, .

Initial studies on this problem have considered specific type of monotone objective functions known as submodular functions. For a given finite set $V = \{v_1, \cdots, v_n\}$, set function $f : 2^V \to \mathbb{R}$ is submodular if and only if any of the following equivalent conditions hold:

1. For any subsets $X \subseteq Y \subseteq V$ and item $v \notin Y$, we have

$$f(X \cup \{v\}) - f(X) \geq f(Y \cup \{v\}) - f(Y).$$

2. For any subsets $X, Y \subseteq V$, we have

$$f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y).$$

3. For any subset $X \subseteq V$ and items $v_1 \neq v_2 \notin X$, we have

$$f(X \cup \{v_1\}) + f(X \cup \{v_2\}) \geq f(X \cup \{v_1, v_2\}) + f(X).$$

The area of submodular function optimisation under given static constraints has been studied quite extensively in the literature. Nemhauser et al. considered maximising nondecreasing submodular functions under a cardinality constraint [NW81]. They proved that the greedy algorithm which iteratively adds a new element with a maximum marginal gain could maintain $(1 - 1/e)$-approximation. Following their results, Khuller et al. showed the unbounded approximation of the greedy approach

in maximising submodular functions under linear constraints [KMN99]. Generalising the greedy approach using a simple modification, however, they showed that it could guarantee optimal $(1/2)(1-1/e)$-approximation in particular submodular functions for the maximum coverage problem. The generalised greedy algorithm compares the result of the iteratively found solution with the best feasible single-element solution and returns the maximum. Krause and Guestrin extended $(1/2)(1-1/e)$ approximation ratio to maximising submodular functions under linear constraint [KG05], which later has been improved to $(1-1/\sqrt{e})$ by Lin and Bilmes [LB10].

In the case of monotone submodular functions, greedy algorithms are often able to achieve the best possible worst-case approximation guarantee (unless $\mathcal{P} = \mathcal{NP}$). Motivated by many real-world applications, the performance of greedy algorithms in general cost functions have also been considered later. Iyer and Bilmes studied cases in which the cost function is monotone and submodular [IB13], while Zhang and Vorobeychik investigated problems with only monotone cost functions [ZV16] as a more general case.

To aid consideration of the general problems, some new definitions have been presented. Submodularity ratio [ZV16], for example, is a definition that we use in Chapter 8; it determines how close a function is to being submodular. The function f is α_f-submodular where

$$\alpha_f = \min_{X \subseteq Y, v \notin Y} \frac{f(X \cup v) - f(X)}{f(Y \cup v) - f(Y)}.$$

This definition is equivalent to the definitions of submodularity when $\alpha_f = 1$, i. e. we have submodularity in this case.

Another notation which is used in our analysis is the curvature of function f. The curvature measures deviation from linearity and reflects the effect of marginal contribution according to the function f [CC84; Von10]. For a monotone submodular function $f : 2^V \to \mathbb{R}^+$,

$$\kappa_f = 1 - \min_{v \in V} \frac{f(V) - f(V \setminus v)}{f(v)}$$

is defined as the total curvature of f.

3.9 Conclusion

In this chapter, we presented definitions of the problems that have been considered throughout the rest of this thesis. We first introduced the definition of linear pseudo-Boolean functions and a well-known specific case of the OneMax problem. Moreover, we described two $\mathcal{NP}$-hard packing problems: the knapsack problem and its

more complicated extension, which is known as the packing while travelling problem.

Next, we presented our graph problems: the minimum spanning tree, which has been considered in Chapter 5, and vertex cover problem alongside its dynamic version which is reviewed Section 4.2.3. Finally, after presenting the makespan scheduling problem, we defined a more general type of problem, known as subset selection, which includes all the problems considered in this thesis.

Chapter 4

A Survey on Evolutionary Algorithms in Dynamic Environments

4.1 Introduction

Real-world problems are often stochastic and have dynamic components. Evolutionary algorithms and other bio-inspired algorithmic approaches such as ant colony optimisation have been applied to a wide range of dynamic problems. The goal of this chapter is to give an overview on recent theoretical developments in the area of evolutionary computation for dynamic problems in the context of discrete optimisation.

Dynamic problems constitute an important part occurring in real-world applications. Problems can change over time due to different components becoming unavailable or available at a later point in time. Different parts of the problem that can be subject to a change are the objective function and possible constraints of the given problem. In terms of scheduling of trains, trains might become unavailable due to mechanical failures and it might be necessary to reschedule the trains in the network in order to still serve the demands of the customers well.

The area of runtime analysis has contributed many interesting studies to the theoretical understanding of bio-inspired algorithms in this area. We start by investigating popular benchmark algorithms such as randomised local search (RLS) and $(1+1)$ EA, which have been introduced in Sections 2.2.1 and 2.3.6, on different dynamic problems. This includes dynamic versions of ONEMAX, the classical vertex cover problem, the makespan scheduling problem, and problem classes of the well-known knapsack problem.

Ant colony optimisation (ACO) algorithms are the other important type of bio-inspired algorithms that has been used and analysed for solving dynamic and stochastic problems. Due to their different way of constructing solutions, based on sampling from the underlying search space by performing random walks on a so-called

construction graph, they have a different ability to deal with dynamic problems. Furthermore, an important parameter in ACO algorithms is the pheromone update strength which allows to determine how quickly previously good solutions are forgotten by the algorithms. This parameter plays a crucial role when distinguishing ACO algorithms from classical evolutionary algorithms. At the end of this chapter, we present a summary of the obtained results on the dynamic problems in the context of ACO.

The contents of this chapter are based on the chapter seven of the book "Theory of Evolutionary Computation" [DN20; NPR20]. This chapter is organised as follows. We present the results obtained for evolutionary algorithms in Section 4.2. We highlight theoretical results on the behaviour of ACO algorithms for dynamic problems in Section 4.3, and finish with some conclusions.

4.2 Analysis of Evolutionary Algorithms on Dynamic Problems

In this section, we summarise recent theoretical analyses that have been performed on evolutionary algorithms dealing with dynamic optimisation problems.

In dynamically changing optimisation problems, some part of the problem is subject to change over time. Usually changes to the objective function or the constraints of the given problem are considered. The different problems that have been studied from a theoretical perspective will be introduced in the forthcoming subsections.

The theoretical analysis of evolutionary algorithms for dynamic problems concentrates on the classical algorithms such as randomised local search (RLS) and $(1+1)$ EA. Analysing evolutionary algorithms with respect to their runtime behaviour, one considers the number of solutions that are produced until a solution of desired quality has been achieved. The expected time to reach this goal refers to the expected number of such solutions. The *expected optimisation time* refers to the expected number of solutions that are produced until an optimal search point has been produced for the first time. Considering dynamic problems, we are often interested in the *expected re-optimisation time* of an algorithm. Starting with a good (or even optimal) solution for the considered problem, the expected number of constructed solutions required to obtain a solution of the same quality after a dynamic change has occurred is analysed.

Shi et al. have investigated the efficiency of evolutionary algorithms for solving linear pseudo-Boolean functions with a dynamic linear constraint in [Shi+19]. Particular attention has been paid to the ONEMAX problem. ONEMAX has been the centre of attention in some other related works as well [Dro02; KLW15]. We first present the investigations that have been performed on this problem, then we give a summary of the results that have been obtained for linear pseudo-boolean functions under

$(1+1)$ **EA**	**MOEA**	**MOEA-S**	**MOGA**	
$O\left(n \log\left(\frac{n-B}{n-B^*}\right)\right)$	$O\left(nD \log\left(\frac{n-B}{n-B^*}\right)\right)$	$O\left(n \log\left(\frac{n-B}{n-B^*}\right)\right)$	$O\left(\min\{\sqrt{n}D^{\frac{3}{2}}, D^2\sqrt{\frac{n}{n-B^*}}\}\right)$	if $B < B^*$
$O\left(n \log\left(\frac{B}{B^*}\right)\right)$	$O\left(nD \log\left(\frac{B}{B^*}\right)\right)$	$O\left(n \log\left(\frac{B}{B^*}\right)\right)$	$O\left(\min\{\sqrt{n}D^{\frac{3}{2}}, D^2\sqrt{\frac{n}{B^*}}\}\right)$	if $B > B^*$

TABLE 4.1: Upper bounds on the expected re-optimisation times of evolutionary algorithms on the ONEMAX problem with a dynamic uniform constraint.

dynamic uniform constraints. Furthermore, in this section we explain the analysis that has been carried out for the dynamic vertex cover problem and the dynamic makespan scheduling problem. Another problem which has been investigated in the context of dynamic optimisation is the MAZE problem for which evolutionary algorithms as well as ant colony optimisation algorithms have been theoretically studied [KM12; LW16; LW17]. The results of evolutionary algorithms and ACO algorithms for this problem are presented in Section 4.2.5 and Section 4.3, respectively.

4.2.1 ONEMAX Under Dynamic Uniform Constraints

The first runtime analysis of evolutionary algorithms for a dynamic discrete problem has been presented by Droste [Dro02]. In that article, the ONEMAX problem is considered and the goal is to find a solution which has the minimum Hamming distance to an objective bit-string. A dynamic change in that work is changing one bit of the objective bit-string, which happens at each time step with probability p' and results in the dynamic changes of the fitness function over time. Droste has found the maximum rate of the dynamic changes such that the expected optimisation time of $(1+1)$ EA remains polynomial for the studied problem. More precisely, he has proved that $(1+1)$ EA has a polynomial expected runtime if $p' = O(\log(n)/n)$, while for every substantially larger probability the runtime becomes super polynomial. It is worth noting that the results of that article hold even if the expected re-optimisation time of the problem is larger than the expected time until the next dynamic change happens.

Using drift analysis, Kötzing et al. [KLW15] have reproved some of the results in [Dro02]. Furthermore, they have carried out theoretical investigations for the extended dynamic ONEMAX problem (see Section 3.2.1), in which each variable can take on more than two values. They also carried out an *anytime analysis* (introduced in [JZ14]) and show how closely their investigated algorithm can track the dynamically moving target over time.

The optimisation time of evolutionary algorithms for ONEMAX and the general class of linear pseudo-Boolean function, under a dynamic uniform constraint given in Section 3.2 has been analysed in [Shi+19]. For now, we concentrate on ONEMAX with with dynamic uniform constraint. The authors have analysed a standard $(1+1)$ EA (Algorithm 3) and three other evolutionary algorithms which are presented

Algorithm 7: MOEA; Assuming $B \leq B^*$. [Shi+19]

```
P ← an initial solution;
while stopping criterion not met do
    Choose x ∈ P uniformly at random;
    Obtain y from x by flipping each bit of x with probability 1/n;
    if (B* ≥ |y|_1 ≥ B) ∧ (∄w ∈ P: w ≽_MOEA y) then
        P ← (P ∪ {y}) \ {z ∈ P | y ≻_MOEA z};
```

in Algorithms 7 to 9. The results of their investigations are summarised in Table 4.1. The $(1+1)$ EA analysed in this paper, uses the fitness function

$$f_{(1+1)}(x) = f(x) - (n+1) \cdot \max\{0, |x|_1 - B^*\}$$

already introduced in [Fri+17]. It gives a large penalty to infeasible solutions by subtracting for each unit of constraint violation a term of $(n+1)$. This implies that each infeasible solution is worse than any feasible one. The penalty of this fitness function, guides the search towards the feasible region and does not allow the $(1+1)$ EA to accept an infeasible solution after a feasible solution has been found for the first time.

Shi et al. [Shi+19] have used multiplicative drift analysis [DJW12] for investigating the behaviour of the studied algorithms. The potential function that they have used for analysing $(1+1)$ EA on ONEMAX with a dynamic uniform constraint is $|x|_0$, when $B \leq B^*$. Here, the initial solution, denoted by x_{org}, is feasible, and the algorithm needs to increase the number of ones of the solution, until the cardinality bound B^* is reached. In this situation, the drift on $|x|_0$ is $\Omega(|x|_0/n)$ for $(1+1)$ EA. Using multiplicative drift analysis, the expected number of generations to reach a solution x^* with $|x^*|_0 = n - B^*$ is

$$\mathrm{O}\left(n \log\left(\frac{|x_{\mathrm{org}}|_0}{|x^*|_0}\right)\right) = \mathrm{O}\left(n \log\left(\frac{n-B}{n-B^*}\right)\right).$$

For the situation where $B \geq B^*$, the initial solution is infeasible and the number of ones of the solution need to decrease (and possibly increase again, in case the last move to the feasible region has decreased $|x|_1$ to less than B^*). The considered potential function in this situation is $|x|_1$ and the drift on that is $\Omega(|x|_1/n)$, giving an expected re-optimisation time of $\mathrm{O}\left(n \log\left(\frac{B}{B^*}\right)\right)$.

The second algorithm that the authors have investigated is the Multi-Objective Evolutionary Algorithm (MOEA) (see Algorithm 7). Here dominance of solutions is defined with respect to the vector-valued fitness function

$$f_{\mathrm{MOEA}}(x) = (|x|_1, f(x)).$$

Algorithm 8: MOEA-S; Assuming $B \leq B^*$. [Shi+19]

```
P ← an initial solution;
while stopping criterion not met do
    Choose x ∈ P uniformly at random;
    Obtain y from x by flipping bit one bit x_i, i ∈ {1, ..., n} chosen u.a.r.;
    if ∀z ∈ P: y ||_{MOEA−S} z then
        P ← P ∪ {y}
    if (B* ≥ |y|_1 ≥ B) ∧ (∃z ∈ P: y ≽_{MOEA-S} z) then
        z ← y;
```

A solution y dominates a solution z w. r. t. f_{MOEA} ($y \succeq z$) iff $|y|_1 = |z|_1$ and $f(y) \geq f(z)$. Furthermore, y strictly dominates z ($y \succ z$) iff $y \succeq z$ and $f(y) > f(z)$. The algorithm keeps at most one individual for each Hamming weight between B and B^*. Let $D = |B^* - B|$, then the size of population P is at most $D + 1$. The analysis shows that this population size slows down the re-optimisation process for the ONEMAX problem. For the case where $B < B^*$ and $B > B^*$, the potential function that Shi et al. [Shi+19] have used for analysing this algorithm is $M = \min_{x \in P} |x|_0$ and $M = \max_{x \in P} |x|_1$, respectively. The analysis is similar to their analysis of $(1+1)$ EA, except that the drift on M is $\Omega(\frac{M}{n \cdot D})$. The D in the denominator comes from the fact that selecting the individual x with minimum $|x|_0$ for $B < B^*$ (minimum $|x|_1$ for $B > B^*$) from the population, happens at each iteration with probability at least $\frac{1}{D+1}$. Using multiplicative drift analysis, they obtained an upper bound of $\mathrm{O}\big(nD \log\big(\frac{n-B}{n-B^*}\big)\big)$ for $B < B^*$ and an upper bound of $\mathrm{O}\big(nD \log\big(\frac{B}{B^*}\big)\big)$ for $B > B^*$.

The third investigated algorithm is a variant of MOEA named MOEA-S shown in Algorithm 8. In this algorithm only single-bit flips are allowed and a different definition for dominance is used. The new notion of dominance does not let the population size grow to a size larger than 2. If $B \leq B^*$, for two bit strings $y, z \in \{0,1\}^n$ we have:

- y dominates z, denoted by $y \succcurlyeq_{\text{MOEA}-\text{S}} z$ if *at most one* value among $|y|_1$ and $|z|_1$ equals B^* or $B^* - 1$, and $(|y|_1 > |z|_1) \vee (|y|_1 = |z|_1 \wedge f(y) \geq f(z))$
- y dominates z, denoted by $y \succcurlyeq_{\text{MOEA}-\text{S}} z$ if both $|y|_1, |z|_1 \in \{B^*, B^* - 1\}$, and $|y|_1 = |z|_1 \wedge f(y) \geq f(z)$

This implies that y and z are incomparable, denoted by $y \parallel_{\text{MOEA}-\text{S}} z$, iff $|y|_1 = B^*$ and $|z|_1 = B^* - 1$ or vice versa.

For $B > B^*$, a similar definition of dominance is given by switching the dependency of $|y|_1 \geq |z|_1$ on the number of 1-bits to $|y|_1 \leq |z|_1$. The results of MOEA-S are obtained by observing that this algorithm behaves like RLS on ONEMAX. It is shown that the expected re-optimisation time for ONEMAX with a dynamic uniform constraint is $\mathrm{O}\big(n \log\big(\frac{n-B}{n-B^*}\big)\big)$ if $B < B^*$ and $\mathrm{O}\big(n \log\big(\frac{B}{B^*}\big)\big)$ if $B > B^*$.

Algorithm 9: MOGA; Assuming $B \leq B^*$ [Shi+19], Concept from [DDE15].

$P \leftarrow \{x\}$, x an initial solution;
while *stopping criterion not met* **do**
  `/* Mutation phase. */`
  Choose $x \in P$ uniformly at random;
  Choose ℓ according to $\mathrm{Bin}(n, p)$;
  for $i = 1$ *to* λ **do**
    $x^{(i)} \leftarrow \mathrm{mutate}_\ell(x)$;
  $V = \{x^{(i)} \mid x^{(i)} \text{ is valid}\}$;
  if $V \neq \emptyset$ **then**
    Choose $x' \in V$ uniformly at random;
  else $x' \leftarrow x$;
  `/* Crossover phase. */`
  for $i = 1$ *to* λ **do**
    $y^{(i)} \leftarrow \mathrm{cross}_c(x, x')$;
  $M = \{y^{(i)} \mid y^{(i)} \text{ is } \succcurlyeq_{\mathrm{MOEA}}\text{-maximal} \wedge |y^{(i)}|_1 = |x|_1 + 1\}$;
  if $M = \{y\}$ **then**
    $y' \leftarrow y$;
  else $y' \leftarrow x$;
  `/* Selection phase. */`
  if $(B^* \geq |y'|_1 \geq B) \wedge (\nexists w \in P \colon w \succcurlyeq_{\mathrm{MOEA}} y')$ **then**
    $P \leftarrow (S \cup \{y'\}) \setminus \{z \in S \mid y' \succ_{\mathrm{MOEA}} z\}$;

Shi et al. [Shi+19] have also introduced a multi-objective variant of the $(1 + (\lambda + \lambda))$ GA [DDE15], which is the fourth algorithm that they have analysed for OneMax with a dynamically changing uniform constraint. In this algorithm, the same notion of dominance as MOEA is used, and the population size can grow to $D + 1$. Having a solution x, at each iteration λ offspring are generated by the mutation operator, which flips $l = \mathrm{Bin}(n, p)$ random bits of x, where p is the mutation probability. The offspring that have a 0 flipped to 1 (a 1 flipped to 0) are considered to be valid for $B^* > B$ (for $B^* < B$). One of the valid offspring (if exists), x', is then used in the crossover phase, in which it is recombined with the parent solution λ times. For a crossover probability c, the crossover operator creates a bit-string $y = y_1 y_2 \cdots y_n$, where each bit $y_i, 1 \leq i \leq n$ is chosen to be x_i with probability c, and x'_i otherwise. The algorithm selects the best solution y with Hamming weight one larger than the Hamming weight of x. The solution y is added to the population if it meets the cardinality constraint and is not dominated by any other solution in the population.

It is proved that this algorithm solves the ONEMAX problem with a dynamically changing uniform constraint in expected time

$$O\left(\min\left\{\sqrt{n}D^{\frac{3}{2}}, D^2\sqrt{\frac{n}{n-B^*}}\right\}\right)$$

if $p = \frac{\lambda}{n}$, $c = \frac{1}{\lambda}$, $\lambda = \sqrt{n/(n-|x|_1)}$ for $B^* > B$, and in expected time

$$O\left(\min\left\{\sqrt{n}D^{\frac{3}{2}}, D^2\sqrt{\frac{n}{B^*}}\right\}\right)$$

if $\lambda = \sqrt{n/|x|_1}$ for $B^* < B$ [Shi+19]. The key argument behind these results is to show a constant probability of producing a valid offspring in the mutation phase, and then show a constant probability of generating a solution y in the crossover phase that is the same as x except for one bit, which is flipped from 0 to 1 for $B^* > B$ and from 1 to 0 for $B^* < B$.

4.2.2 Linear Pseudo-Boolean Functions Under Dynamic Uniform Constraints

The classical $(1+1)$ EA and three multi-objective evolutionary algorithms have been investigated in [Shi+19] for re-optimising linear functions under dynamic uniform constraints. The general class of linear constraints on linear problems leads to exponential optimisation times for many evolutionary algorithms [Fri+17; ZH07]. Shi et al. [Shi+19] considered the dynamic setting given in Section 3.2 and analyse the expected re-optimisation time of the investigated evolutionary algorithms. This section includes the results that they have obtained, in addition to the proof ideas of their work.

The algorithms that are investigated in their work, are presented in Algorithms 3 of Section 2.3.6 and 7 to 9 of Section 4.2.1; and the results are summarised in Table 4.2. The $(1+1)$ EA (Algorithm 3) uses the following fitness function which has been introduced by Friedrich et al. [Fri+17] (similar to the fitness function for ONEMAX in Section 4.2.1):

$$f_{(1+1)}(x) = f(x) - (n\,w_{\max} + 1) \cdot \max\{0, |x|_1 - B^*\}$$

Here, $w_{\max} = \max_{1\le i\le n} w_i$ denotes the maximum weight, and the large penalty for constraint violations guides the search towards the feasible region.

Shi et al. [Shi+19] have investigated this setting similar to the analysis of ONEMAX under dynamic uniform constraints (Section 4.2.1). The main difference is that for a non-optimal solution with B^* 1-bits, an improvement is not possible by flipping a single bit. A 2-bit flip that flips a 1 and a 0 may be required, resulting in an expected re-optimisation time of $O\big(n^2 \log(B^* w_{\max})\big)$.

$(1+1)$ **EA**	**MOEA**	**MOEA-S**	**MOGA**
$O(n^2 \log(B^* w_{\max}))$	$O(nD^2)$	$O(n \log D)$	$O(nD^2)$

TABLE 4.2: Upper bounds on the expected re-optimisation time of evolutionary algorithms on linear functions with a dynamic uniform constraint.

The second investigated algorithm, MOEA, uses the fitness function f_{MOEA} and the notion of dominance defined in Section 4.2.1. Unlike the re-optimisation time of this algorithm for the ONEMAX problem, whose upper bound is worse than the upper bound of $(1+1)$ EA; for the general linear functions the upper bounds obtained for MOEA are smaller than the ones obtained for $(1+1)$ EA. The reason is that the algorithm is allowed to keep one individual for each Hamming weight between the two bounds in the population. This avoids the necessity for a 2-bit flip. To reach a solution that is optimal for cardinality $A+1$, the algorithm can use the individual that is optimal for cardinality A and flip the 0-bit whose weight is maximal. This happens in an expected number of at most $en(D+1)$ iterations, where $D = |B^* - B|$. As there are $D+1$ different cardinality values between the two bounds, the expected time to reach the optimal solution with cardinality B^* is $O(nD^2)$.

MOEA-S (Algorithm 8) has also been analysed for linear functions with a dynamically changing uniform constraint. It uses single bit-flips and the population includes at most 2 solutions: one with Hamming weight at most $B^* - 1$ and one with Hamming weight B^*. With this setting, long waiting times for selecting a certain individual of the population are avoided. The algorithm starts with one solution in the population. It has been shown that in time $O(n \log D)$ the population consists of one solution with Hamming weight $B^* - 1$ and one with Hamming weight B^*. Then the authors use a potential function to measure the difference of the current population to an optimal solution with Hamming weight B^*. The potential is given by the number of 0-bits in the two solutions that need to be set to 1 in order to obtain an optimal solution. Using multiplicative drift analysis with respect to the potential function, they have proved that the expected re-optimisation time of the algorithm is $O(n \log D)$.

The fourth algorithm that is analysed in [Shi+19] is MOGA (Algorithm 9 of Section 4.2.1). The authors have shown that, choosing an optimal solution of Hamming weight $A < B^*$ for reproduction, an optimal solution for Hamming weight $A+1$ is produced with probability $\Omega(n^{-1/2})$ in the next generation, if $p = \frac{\lambda}{n}$, $c = \frac{1}{\lambda}$ and $\lambda = \sqrt{n}$. Since there are $D+1$ different Hamming weights to consider, and each iteration of the algorithm constructs $O(\lambda) = O(\sqrt{n})$ solutions, the expected re-optimisation time is upper bounded by $O(nD^2)$.

4.2.3 Dynamic Vertex Cover Problem

The common representation for solving the vertex cover problem by means of evolutionary algorithms is the node-based representation [Fri+10; KN13; OHY09; PSN16]. A different representation, the edge-based representation, has been suggested and analysed in [JOZ13] for the static vertex cover problem. In this representation a search point is a bit-string $x \in \{0,1\}^m$, where m denotes the number of edges in the given graph $G = (V, E)$. For a given search point x, $E(x) = \{e_i \in E \mid x_i = 1\}$ is the set of chosen edges. The cover set induced by x, denoted by $V_C(x)$, is the set of all nodes that are adjacent to at least one edge in $E(x)$.

Three variants of RLS and $(1+1)$ EA have been investigated. This includes one node-based approach and two edge-based approaches. The node-based approach and one of the edge-based approaches use a standard fitness function,

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

in which each uncovered edge obtains a large penalty of $|V|+1$. In [JOZ13], an exponential lower bound for finding a 2-approximate solution for the static vertex cover problem with these two approaches using the fitness function f has been shown. Furthermore, considering the dynamic vertex cover problem, Pourhassan et al. [PGN15] have proved that there exist classes of instances of bipartite graphs where dynamic changes on the graph lead to a bad approximation behaviour.

The third variant of an evolutionary algorithm that Jansen et al. [JOZ13] have investigated, is an edge-based approach with a specific fitness function. The fitness function f_e has a very large penalty for common nodes among selected edges. It is defined as

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

This fitness function guides the search towards a matching, and afterwards to a maximal matching. In other words, whenever the algorithms find a matching, then they do not accept a solution that is not a matching, and whenever they find a matching that induces a node set with k uncovered edges, then they do not accept a solution with $k' > k$ uncovered edges. It is well known that taking all the nodes belonging to the edges of a maximal matching for a given graph results in a 2-approximate for the vertex cover problem.

The variant of RLS and $(1+1)$ EA work with the edge-based representation and the fitness function f_e. Note that search points are bit-strings of size m, and the probability of flipping each bit in $(1+1)$ EA is $1/m$. Jansen et al. [JOZ13] have proved that RLS and $(1+1)$ EA with the edge-based approach find a maximal matching which induces a 2-approximate solution for the vertex cover problem in expected time $O(m \log m)$, where m is the number of edges.

The behaviour of RLS and $(1+1)$ EA with this edge-based approach has been investigated on the dynamic vertex cover problem (see Section 3.6) in [PGN15]. It is proved in [PGN15] that starting from a 2-approximate solution for a current instance of the problem, in expected time $O(m)$ RLS finds a 2-approximate solution after a dynamic change of adding or deleting an edge. The authors of that paper have investigated the situation for adding an edge and removing an edge separately. For adding an edge, they have shown that the new edge is either already covered and the maximal matching stays a maximal matching, or it is not covered by the current edge set and the current edge set is a matching that induces a solution with one (only the new edge) uncovered edge. Since the number of uncovered edges does not grow in this approach and the algorithm selects the only uncovered edge with probability $1/m$, a maximal matching is found in expected m steps. This argument also holds for $(1+1)$ EA, but the probability of selecting the uncovered edge and having no other mutations with this algorithm is at least $1/(em)$. Therefore, the expected re-optimisation time for $(1+1)$ EA after a dynamic addition is also $O(m)$.

When an edge is deleted from the graph, if it had been selected in the solution, a number of edges can be uncovered in the new situation. All these uncovered edges had been covered by the two nodes of the removed edge, and can be partitioned into two sets U_1 and U_2, such that all edges of each set share a node. Therefore, if the algorithm selects one edge from each set (if any exist), the induced node set becomes a vertex cover again. It will again be a maximal matching and therefore a 2-approximate solution. On the other hand, no other one-bit flips in this situation can be accepted, because they either increase the number of uncovered edges, or make the solution become a non-matching. With RLS, in which only one-bit flips are possible, the probabilities of selecting one edge from U_1 and U_2 at each step are $\frac{|U_1|}{m}$ and $\frac{|U_2|}{m}$, respectively. Therefore, in expected time $O(m)$ one edge from each set is selected by the algorithm.

The analysis for $(1+1)$ EA dealing with a dynamic deletion is more complicated, because multiple-bit flips can happen. In other words, it is possible to deselect an edge and uncover some edges at the same step where an edge from U_1 or U_2 is being selected to cover some other edges. An upper bound of $O(m \log m)$ is shown in [PGN15] for the expected re-optimisation time for $(1+1)$ EA after a dynamic deletion, which is the same as the expected time to find a 2-approximate solution with that algorithm, starting from an arbitrary solution.

4.2.4 Dynamic Makespan Scheduling

Makespan scheduling is another problem which has been considered in a dynamic setting [NW15]. It is assumed that the processing time of job i for $1 \leq i \leq n$, is $p_i \in [L, U]$, where L and U are lower and upper bounds on the processing time of jobs respectively. In addition, the ratio between the upper bound and the lower bound is denoted by $R = U/L$. The runtime performance of $(1+1)$ EA and RLS is studied in terms of finding a solution with a good discrepancy and it is assumed that there is no stopping criteria for the algorithms except achieving such a solution. The discrepancy $d(x)$ of a solution x is defined as

$$d(x) = \left| \left(\sum_{i=1}^{n} p_i(1 - x_i) \right) - \left(\sum_{i=1}^{n} p_i x_i \right) \right|.$$

Note that a solution that has a smaller discrepancy also has a smaller makespan. Moreover, the proofs benefit from an important observation about the fuller machine (the machine which is loaded heavier and determines the makespan). The observation is on the minimum number of jobs of the fuller machine in terms of U and L. :

- Every solution has at least $\lceil (P/2)/U \rceil \geq \lceil (nL/2)/U \rceil = \lceil (n/2)(L/U) \rceil = \lceil (n/2)R^{-1}) \rceil$ jobs on the fuller machine, where $P = \sum_{i=1}^{n} p_i$

Two dynamic settings are studied for this problem. The first one is called the *adversary model* in which a strong adversary is allowed to choose at most one arbitrary job i in each iteration and change its processing time to a new $p_i \in [L, U]$. It is proven that, independently of initial solution and the number of changes made by the adversary, RLS obtains a solution with discrepancy at most U in expected time of $O(n \min\{\log n, \log R\})$. In the case of RLS, the number of jobs on the fuller machine increases only when the fuller machine is switched. Otherwise it increases the makespan and will not be accepted by the algorithm. This fact is the base of the proof. It is proved that if the fuller machine switches (either by an RLS step, which moves a single job between machines, or by a change that the adversary makes), then a solution with discrepancy at most U has been found in a step before and after the switch.

The proof for $(1+1)$ EA is not as straightforward as for RLS, since $(1+1)$ EA may switch multiple jobs between the machines in one mutation step. However, it is shown that the number of incorrect jobs on the fuller machine, which should be placed on the other machine to decrease the makespan, has a drift towards zero. Using this argument, it is shown that $(1+1)$ EA will find a solution with discrepancy at most U in expected time $O(n^{3/2})$. Whether the better upper bounds such as $O(n \log n)$ are possible is still an open problem.

In the same dynamic setting, recovering a discrepancy of at most U is also studied

for both RLS and $(1+1)$ EA algorithms. It is assumed that the algorithm has already achieved or has been initialised by a solution with the discrepancy of at most U and the processing time of a job changes afterwards. By applying the multiplicative drift theorem on the changes of the discrepancy and using the fact that the discrepancy will change by at most $U - L$, it is proven that $(1+1)$ EA and RLS recover a solution with discrepancy of at most U in expected time of $O(\min\{R, n\})$.

The makespan scheduling problem has also been studied in another dynamic setting. In this model which is called the *random model*, it is assumed that all job sizes are in $\{1, \ldots, n\}$. At each dynamic change, the adversary chooses one job i and its value will change from p_i to $p_i - 1$ or $p_i + 1$ each with probability of $1/2$. The only exceptions are $p_i = n$ and $p_i = 1$ for which it changes to $p_i = n - 1$ and $p_i = 2$, respectively. Overall, this setting has less adversarial power than the *adversary model* due to the randomness and changes by only 1 involved.

Let the random variable X_i denote the random processing time of job i at any point of time. The following lemma proves that no large gap exists in the value of processing times which are randomly chosen for jobs.

Lemma 4.1 (Lemma 4 in [NW15]). *Let* $\phi(i) := |\{X_j \mid X_j = i \wedge j \in \{1, \ldots, n\}\}|$ *where* $i \in \{1, \ldots, n\}$*, be the frequency of jobs of size i. Let*

$$G := \max\{l \mid \exists i : \phi(i) = \phi(i+1) = \cdots = \phi(i+l) = 0\}$$

is the maximum gap size, i. e. maximum number of intervals with zero frequency everywhere. Then, for some constant $c > 0$*,*

$$\Pr(G \geq l) \leq n2^{-cl}.$$

This lemma states that, for any constant $c > 0$ and gap size $G \geq c' \log n$ with a sufficiently large c', there is no gap of size G with probability at least $1 - n \cdot n^{-c-1} = 1 - n^{-c}$. This probability is counted as a high probability in this study.

When the discrepancy is larger than G, it is proven that it decreases by at least one if two jobs swap between the fuller and the emptier machines. Furthermore, the maximum possible discrepancy for an initial solution is n^2 when all the jobs have the processing time of n and are placed on one machine. Finally, it is proven that regardless of the initial solution, $(1+1)$ EA obtains with high probability a discrepancy of at most $O(\log n)$ after a one-time change in time $O(n^4 \log n)$.

The previous result considered the worst-case initial solution. However, it is proven that if the initial solution is generated randomly, then its expected discrepancy is $\Theta(n\sqrt{n})$ and it is $O(n\sqrt{n}\log n)$ with high probability. Thus, with a random initial solution, $(1+1)$ EA obtains a discrepancy of $O(\log n)$ after a one-time change in time $O(n^{3.5}\log^2 n)$ with high probability.

The two results on $(1+1)$ EA and in the random model are for a one-time change. In extreme case, however, the processing time of a job may increase or decrease by one in each step which makes it hard to obtain a discrepancy of $O(\log n)$, unlike the other results in this setting. Although, by using the results in the adversary model and considering that $R = U = n$, it is possible to find a solution with discrepancy of at most n. In the final theorem of this study, it is proven that independently of the initial solution and the number of changes, $(1+1)$ EA and RLS obtain a solution with discrepancy of at most n in expected time $O(n^{3/2})$ and $O(n \log n)$, respectively. In addition, it is shown that the expected ratio between the discrepancy and the makespan is $6/n$. This is done by considering that a solution of discrepancy at most n is obtained together with a lower bound on the makespan. The expected sum of all processing times is $n(n+1)/2$ and it is at least $n^2/3 + n$ with the probability of $1 - 2^{-\Omega(n)}$. Hence, the expected makespan is at least $n^2/6 + n/2$. Furthermore, if the sum of processing times is less than $n^2/3 + n$, then the ratio would be at least n/n since the processing times are at least one. Hence, if n is not too small the ratio is bounded from above by

$$\frac{6}{n} - \frac{3}{n} + 2^{-\Omega(n)} \leq \frac{6}{n}.$$

4.2.5 The MAZE Problem

The dynamic pseudo-boolean function MAZE proposed in [KM12], consists of $n+1$ phases of $t_0 = kn^3 \log n$ iterations. During the first phase, the function is equivalent to ONEMAX. In the next n phases, all bit-strings except two, still have the value equivalent to ONEMAX. The two different bit-strings, for each phase p are 0^p1^{n-p} and $0^{p-1}1^{n-p+1}$ which have fitness values with an oscillating pattern: for two iterations out of three, these two bit-strings are assigned values $n+2$ and $n+1$, respectively, and at the third iteration, this assignment is reversed. Note that during the last phase, MAZE behaves similar to TRAP. The formal definition of MAZE follows:

$$\begin{aligned}
\textsc{Maze}(x,t) &= \begin{cases} n+2 & \text{if } t > (n+1)\cdot t_0 \ \wedge x = 0^n \\ n+2 & \text{if } t > t_0 \ \wedge x = \mathrm{OPT}(t) \\ n+1 & \text{if } t > t_0 \ \wedge x = \mathrm{ALT}(t) \\ \textsc{OneMax}(x) & \text{otherwise} \end{cases} \\
\mathrm{OPT}(t) &= \begin{cases} \mathrm{OPT}_{\lfloor t/t_0 \rfloor} & \text{if } t \neq 0 \mod 3 \\ \mathrm{ALT}_{\lfloor t/t_0 \rfloor} & \text{otherwise} \end{cases} \\
\mathrm{ALT}(t) &= \begin{cases} \mathrm{ALT}_{\lfloor t/t_0 \rfloor} & \text{if } t \neq 0 \mod 3 \\ \mathrm{OPT}_{\lfloor t/t_0 \rfloor} & \text{otherwise} \end{cases} \\
\mathrm{OPT}_p &= 0^p1^{n-p} \text{ for } p \leq n \\
\mathrm{ALT}_p &= 0^{p-1}1^{n-p+1} \text{ for } p \leq n.
\end{aligned}$$

While it was shown in [KM12] that a $(1+1)$ EA loses track of the optimum for this problem and requires with high probability an exponential amount of time to find the optimum, Lissovoi and Witt [LW16] have proved that the optimum of the MAZE function extended to finite alphabets, can be tracked by a $(\mu+1)$ EA when the parent population size μ is chosen appropriately and a genotype diversity mechanism is used.

In another work [LW17], the behaviour of parallel evolutionary algorithms is studied on the MAZE problem. In their analysis, it is proved that both the number of independent sub-populations (or islands), λ, and the length of the migration intervals, τ, influence the results. When τ is small, particularly for $\tau = 1$, migration occurs too often, and the algorithm behaves similar to $(1+\lambda)$ EA and fails to track the MAZE efficiently, for $\lambda = O(n^{1-\varepsilon})$, where ε is an arbitrary small positive constant. But with a proper choice of τ, more precisely $\tau = t_0$, where t_0 is the number of iterations in each phase in the definition of the MAZE problem, and a choice of $\lambda = \Theta(\log n)$, the algorithm is able to track the optimum of MAZE efficiently.

The analysis of $(\mu+1)$ EA and parallel evolutionary algorithms on the MAZE problem shows that both these algorithms have limitations for tracking the optimum. $(\mu+1)$ EA not only exploits the small number of individuals among which the optimum is oscillated, but also requires genotype diversity and a proper choice of μ. On the other hand, the obtained positive results of parallel evolutionary algorithms on the MAZE problem depend on a careful choice of migration frequency. But on the plus side, with parallel evolutionary algorithms, the problem can be extended to a finite-alphabet version.

4.3 Ant Colony Optimisation

After having investigated evolutionary algorithms for dynamic problems, we now give a summary on the results obtained in the context of ant colony optimisation (ACO). ACO algorithms construct solutions for a given optimisation problem by performing random walks on a so-called construction graph. The construction graph frequently used in pseudo-Boolean optimisation is shown in Figure 4.1. This random walk is influenced by so-called pheromone values on the edges of the graph. At each time step, the pheromone values induce a probability distribution on the underlying search space which is used to sample new solutions for the given problem. Pheromone values are adapted over time such that good components of solutions obtained during the optimisation process are reinforced. The idea is that this reinforcement then leads to better solutions during the optimisation process. An algorithm which is frequently studied throughout theoretical investigations for pseudo-Boolean maximisation is MMAS (see Algorithm 7). It is a simplification of the Max-Min Ant System introduced in [SH00]. The algorithm, which is given in Algorithm 7, only uses one ant in each iteration. However, variants of MMAS called

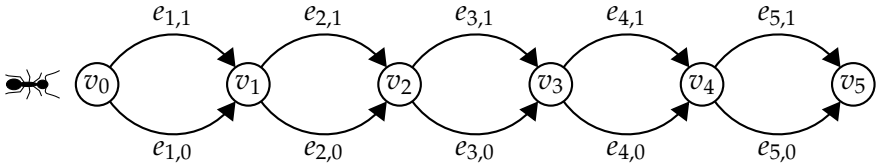


FIGURE 4.1: Construction graph for pseudo-Boolean optimisation with $n = 5$ bits.

λ-MMAS, where in each iteration λ ants are used, have also been studied in the literature. Pheromone values are chosen within the interval $[\tau_{\min}, \tau_{\max}]$ where $\tau_{\min}$ and $\tau_{\max}$ are lower and upper bounds used in MMAS. Furthermore, the update strength ρ plays an important role in the runtime analysis of ACO algorithms. For MMAS, a large update strength such as $\rho = 1$ often makes the considered MMAS algorithms similar to simple evolutionary algorithms such as $(1+1)$ EA. The considered algorithms are usually analysed with respect to the number of solutions until a given goal has been achieved. As in the case of runtime analysis of evolutionary algorithms, one is often interested in the expected number of solutions to reach the desired goal.

Algorithm 10: MMAS

```
Set τ_(u,v) = 1/2 for all (u, v) ∈ E.;
Construct a solution x*.;
Update pheromones w. r. t. x*.;
repeat forever
    Construct a solution x.;
    if f(x) ≥ f(x*) then x* := x.;
    Update pheromones w. r. t. x*.;
```

4.3.1 Dynamic Problems

Kötzing and Molter [KM12] compared the behaviour of $(1+1)$ EA and MMAS on the MAZE problem. The MAZE problem has an oscillating behaviour of different parts of the function and the authors have shown that MMAS is able to track this oscillating behaviour if ρ is chosen appropriately, i.e. $\rho = \theta(1/n)$ whereas $(1+1)$ EA looses track of optimum with probability close to 1.

In the case of dynamic combinatorial optimisation problems, dynamic single-source shortest paths problems have been investigated in [LW15]. Given a destination node $t \in V$, the goal is to compute for any node $v \in V \setminus t$ a shortest paths from v to t. The set of these single-source shortest paths can be represented as a tree with root t and the path from v to t in that tree gives a shortest path from v to t. The authors have investigated different types of dynamic changes for variants of MMAS. They first investigated the MMAS and show that this algorithm can effectively deal with one time changes and build on investigations in [ST12] for the static case. They show that

the algorithm is able to recompute single-source shortest paths in an expected number of $O(\ell^*/\tau_{\min} + \ell \ln(\tau_{\max}/\tau_{\min})/\rho)$ iterations after a one change has happened. The parameter ℓ denotes the maximum number of arcs in any shortest path to node t in the new graph and $\ell^* = \min\{\ell, \log n\}$. The result shows that MMAS is able to track dynamic changes if they are not too frequent. Furthermore, they present a lower bound of $\Omega(\ell/\tau_{\min})$ in the case that $\rho = 1$ holds. Afterwards, periodic local and global changes are investigated. In the case of the investigated local changes, λ-MMAS with a small λ is able to track periodic local changes for a specific setting. For global changes, a setting with oscillation between two simple weight functions is introduced where an exponential number of ants would be required to make sure that an optimal solution is sampled with constant probability in each iteration.

4.4 Conclusions

Evolutionary algorithms have been extensively used to deal with dynamic problems. We have given an overview on recent results regarding the theoretical foundations of evolutionary algorithms for dynamic problems in the context of rigorous runtime analysis. Various results for dynamic problems in the context of combinatorial optimisation for problems such as makespan scheduling and minimum vertex cover have been summarised and the benefits of different approaches to deal with such dynamic problems have been pointed out.

While all these studies have greatly contributed to the understanding of the basic working principles of evolutionary algorithms and ant colony optimisation in dynamic environments, analysing the behaviour on complex problems remains highly open.

Part II

Static Combinatorial Optimisation Problems

Chapter 5

Evolutionary Algorithms with Biased Mutation for the Minimum Spanning Tree Problem

5.1 Introduction

In this chapter, we consider the minimum spanning tree problem which has been described in Section 3.5. Given an undirected edge-weighted graph, the goal is to find a spanning sub-graph which is a tree and has minimal total weight among all such trees. When each edge is assigned multiple – usually conflicting – weights, one is interested in a set of multi-objective compromise solutions (moMST).

Usually evolutionary algorithms perform an unbiased search due to their frequent application in settings where knowledge on the fitness function can only be gained by fitness function evaluations. However, if domain knowledge on the composition of (Pareto-)optimal solutions is available one should incorporate this knowledge into mutation operators to speed up the evolution considerably [DHN07; DHN06; FQW18; Fri+18; JS10].

In this chapter we consider biased mutation for evolutionary algorithms for the single- and multi-objective MST problem and compare with unbiased counterparts. Specifically, we examine the effects of mutation bias on the time complexity of simple EAs until they hit an optimal solution or cover the Pareto-front for the first time. We show that bias can be both boon and bane depending on the structure of optimal solutions on example graphs. I. e., there are situations where introduced bias leads to improved upper bounds where we save a factor of n if the ranks of edges which are part of optimal solutions are $O(n)$. Contrarily, if heavy edges are frequent members of optimal solutions, bias towards lightweight edges may entail an exponential deterioration in the expected running time. Luckily, in the single-objective setting, we can combine the best of both worlds. A simple modification, which decides for unbiased or biased mutation in each step independently with probability $1/2$, leads to a guaranteed polynomial runtime bound of $O(n^3 \log(n \cdot w_{\max}))$ for general graphs

where $w_{\max}$ is the maximum edge weight in the graph. At the same time this strategy benefits from bias if the circumstances allow for it saving on a factor of n.

The contributions of this chapter has been accepted to be presented at the genetic and evolutionary computation conference (GECCO 2020) [RBN20]. We introduce the (mo)MST problem formally, establish a vocabulary and introduce the considered algorithms in Section 5.2. Sections 5.3 and 5.4 deal with our runtime analysis in the single-objective and multi-objective MST setting, respectively. Section 5.5 wraps up the work with some concluding remarks.

5.2 Preliminaries

Let $G = (V, E)$ denote a graph with vertex set V and edge set E. For convenience, we write $n = |V|$ and $m = |E|$. Consider the definition of single objective MST in Section 3.5. In the multi-objective scenario, each edge is assigned two weights $w(e) = (w_1(e), w_2(e))$.[1] The goal is to find a spanning tree such that the total weight in both weight functions is minimised simultaneously. This may result in a set of incomparable trade-offs which are not necessarily better than each other in both weights. In order to capture this aspect mathematically we adopt the well-known notion of Pareto dominance to establish a partial order of spanning trees. Let $w(T_i) = \sum_{e \in T} w_i(e)$ be the weight of spanning tree T_i. We say spanning tree T_1 weakly (Pareto-)dominates spanning tree T_2, denoted by $T_1 \succeq T_2$, if $w_1(T_1) \leq w_1(T_2) \wedge w_2(T_1) \leq w_2(T_2)$. The strong dominance holds when at least one of the inequalities is strict and it is denoted by $T_1 \succ T_2$. T_1 is called non-dominated if there is no other spanning tree that dominates T_1. Likewise, $w(T_1) = (w_1(T_1), w_2(T_1))$ is the non-dominated objective vector. The union set of all non-dominated spanning trees is called Pareto set, its image in objective space is called the Pareto front, and each solution is termed a Pareto(-optimal) solution or multi-objective MST (moMST). Our goal is to find a non-dominated spanning tree for each non-dominated objective vector. In the following, we present the algorithms that we use to tackle these problems.

5.2.1 Algorithms

We consider the performance of a version of (1+1) EA_{MST}, which is designed based on $(1+1)$ EA of Section 2.3.6 to deal with our graph problem (see Algorithm 11), facing the single-objective MST problem. (1+1) EA_{MST} is initialised with a random spanning tree T. There have been different studies on generating random spanning trees such as a rather classical randomised algorithm by Broder [Bro89], with expected running time of $O(n \log n)$ for almost all graphs or more recently by Madry et al. [MST15]. Afterwards, the algorithm sets an *edge-selection strategy*, i. e., the edge-selection probability distribution that is used in Line 6. Next, the algorithm

[1] Clearly, more than two objective functions are possible. Since we restrict our analysis to bi-objective problems in this chapter we refrain from introducing the general form in favour of less notation overhead.

Algorithm 11: (1+1) EA_{MST}

Let T be a random spanning tree on $G = (V, E)$.;
Set the edge-selection strategy;
while *optimum not found* **do**
    $T' \leftarrow T$;
    $k \leftarrow 1 + \text{Pois}(1)$;
    Based on the selection strategy, assign the probability $q(e)$ to each edge $e \in E$.;
    for k *times* **do**
        Choose $e \in E$ with probability $q(e)$.;
        $T' \leftarrow T' \cup \{e\}$;
        Drop an edge from the resulting cycle in T' uniformly at random.;
    if T' *has no worse fitness than* T **then**
        $T \leftarrow T'$;

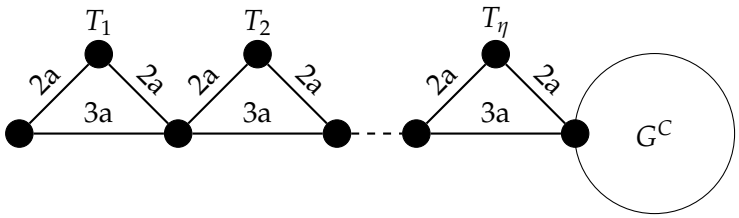


FIGURE 5.1: Triangular-tailed graph G with a chain of $p = n/4$ triangles and a giant component $G^C = K_{n/2}$. [NW07]

sets $k = \text{Pois}(1) + 1$, the number of edges for the mutation step, where $\text{Pois}(1)$ stems from a Poisson distribution with rate $\lambda = 1$. The constant ensures that we always perform at least one mutation and avoids counting iterations that does not generate new solutions. The same approach has been used in [RPN19]. In the mutation step, an edge is selected according to its probability $q(e)$ and is added to T. As the mutant is no longer acyclic after the edge insertion, removing a randomly chosen edge from the unique cycle is required to reestablish the tree property. This guarantees that the resulting graph is a spanning tree. The algorithm repeats this procedure k times to achieve a new solution T' and replaces T by T' if $w(T') \leq w(T)$.

We consider three versions of Algorithm 11 where the difference is in the edge-selection strategy.

(1+1) EA-UM refers to the unbiased variant of (1+1) EA_{MST} in which always each edge is selected with uniform probability $q(e) = 1/m$. We also consider (1+1) EA_{MST} with biased mutation called (1+1) EA-BM, in which the mutation probability of edge e has been set based on the approximation of the probability that e appears in the MST. The approximation, which is the result of experimental analyses, gives higher probability to the edges with lower weights to be selected. The details on how to calculate the approximation is given in the following sections. Note that for these

Algorithm 12: G-SEMO$_{\text{MST}}$

```
1  Initialise population P with a random spanning tree on G = (V, E).;
2  Set the edge-selection strategy.;
3  while not all Pareto-optimal solutions found do
4      Choose T ∈ P uniformly at random.;
5      T' ← T;
6      k ← 1 + Pois(1);
7      Based on the selection strategy, assign the probability q(e) to each edge
         e ∈ E.;
8      for k times do
9          Choose e ∈ E with probability q(e).;
10         T' ← T' ∪ {e};
11         Drop an edge from the resulting cycle in T' uniformly at random.;
12     if {T'' ∈ P | T'' ≻ T'} = ∅ then
13         P = P \ {T'' ∈ P | T' ⪰ T''} ∪ {T'};
```

versions of (1+1) EA$_{\text{MST}}$, the edge-selection strategy does not change during the optimisation process and has been set at the beginning of the algorithm. In other words, the edge-selection strategy deterministically assigns values of $q(e)$ (see line 6 in Algorithm 11), i. e., either uniform or biased mutation with probability 1. Additionally, we analyse a "hybrid" (1+1) EA$_{\text{MST}}$, called (1+1) EA-MM (MM for mixed mutation), where in each iteration of the outer loop the algorithm decides by fair coin-tossing which strategy (biased or unbiased) to use.

For the multi-objective scenario, our runtime analysis is based on the G-SEMO$_{\text{MST}}$ (see Algorithm 12) which is a version of G-SEMO algorithm introduced in Chapter 2. G-SEMO$_{\text{MST}}$ stores a set of non-dominated solutions in the population P, which is initialised with a single random spanning tree. In each iteration, it selects a solution T from P uniformly at random and sets the number of the edges to be added in the mutation step: one plus a random value sampled from a Poisson distribution with $\lambda = 1$. The mutation step is the same as the (1+1) EA$_{\text{MST}}$ and guarantees that the resulting graph T' is also a spanning tree. If there is no solution in P that strongly dominates T', T' is added to P and all the solutions that T' weakly dominates are removed from P. Similar to the single-objective setting, two versions are subject to analysis: G-SEMO-UM with uniform edge-selection probability $q(e) = 1/m$ and its biased counterpart G-SEMO-BM, in which edges that are dominated by fewer edges in E have higher probability to be selected for the mutation (see Section 5.4 for details).

5.3 Single-Objective Problem

In this section, we consider two types of triangular-tailed graphs, G_1 and G_2, which are structurally the same but are different in the weights of the edges. A triangular-tailed graph consists of a clique, G^C, with $\nu = n/2$ vertices and a triangular tail, G^T, with $\eta = n/4$ triangles (Figure 5.1). In both G_1 and G_2, each triangle has 2 edges with weights $2a$ and one edge with weight $3a$, where $a := n^2$. The weights of edges in the clique are $4a$ and a in G_1 and G_2, respectively.

Neumann and Wegener proved that (1+1) EA_{MST}, which uses bit-string representation, flips each bit with probability $1/m$ and is initialised with a random graph, finds the MST of the triangular-tailed graphs in $\Omega(n^4 \log n)$ expected time [NW07], i. e. the triangular-tailed graph has been used as the worst case example to prove the lower bound. This bound is proven for a fitness function that prevents the algorithm to accept solutions other than spanning trees after achieving the first spanning tree. Moreover, the most time consuming phase in their proof is finding the MST from an achieved spanning tree. Hence, their proof also holds even if (1+1) EA_{MST} is initialised with a spanning tree.

Using the same worst case example, we prove that (1+1) EA-UM finds the MST in $\Theta(n^2 \log n)$. Afterwards, we improve this bound for graph G_1, in which the edges of G^T are lighter than the edges of the clique, by enhancing the biased mutation in (1+1) EA-BM. Inspired by the study of Raidl et al. [RKJ06], we use the ranking strategy to perform the biased mutation. To this aim, we assign rank r, $1 \leq r \leq |E|$, to each edge based on its placement in ascending order of the weights, ties are broken uniformly at random. For each edge $e \in E$ with rank r, we approximate the probability of e to appear in the MST with $p(r) = a^r$. Then, we set

$$q(e) = q_{\text{b}}(e) = \frac{\sqrt{p(r)}}{\sum_{i=1}^{m} \sqrt{p(r)}},$$

as the probability of selecting e for the mutation step, where $a = \frac{n-1}{n}$. We show that (1+1) EA-BM finds the MST of G_1 in expected time $\Theta(n \log n)$. However, it takes exponential time for (1+1) EA-BM to find the MST of G_2, in which the edges of G^T are heavier than the edges of G^C. In the following proofs, let $b = B(T)$ denote the set of bad selected edges in the tail of solution T, which have weight $3a$.

Lemma 5.1. *(1+1) EA-BM and (1+1) EA-UM do not increase the value of b during the optimisation process.*

Proof. Let T' be the result of k subsequent edge insertions into T by mutation. Any changes in the structure of the solution in G^C does not change the weight and neither b. It is similar when an edge with weight $2a$ is added and the other edge with weight $2a$ is removed from the cycle. Therefore, we only consider the number of changes in G^T that the swap between $2a$ and $3a$ edges happen in the same triangle. Let b_{i} and b_{d}

denote the number of swaps that increase and decrease $w(T)$, respectively. We have $|B(T')| = |B(T)| + b_i - b_d$ and $w(T') = w(T) + a(b_i - b_d)$. On the other hand, the algorithms accept T' if and only if $w(T') \leq w(T)$, which implies that $b_i \leq b_d$. Thus, in an accepted move, the number of bad edges added to T is less than or equal to the number of added edges with weight $2a$. □

The following theorem considers the performance of (1+1) EA-UM on triangular-tailed graphs.

Theorem 5.2. *(1+1) EA-UM finds the MST of triangular-tailed graph $G \in \{G_1, G_2\}$ in $\Theta(n^2 \log n)$ steps with probability $1 - o(1)$.*

Proof. Here, we follow the proof of Claim 10 in [NW07]. Note that we can focus on G^T since the initial solution is a random spanning tree and all weights in G^C are equal. Moreover, the MST contains all $2a$ edges and no $3a$ edge. Since (1+1) EA-UM does not increase b (Lemma 5.1), we need to calculate the expected time to achieve $b = 0$. In order to reduce b by one, the algorithm needs to insert a $2a$ edge and remove the $3a$ edge from the resulting cycle. The probability of adding only one edge is the probability of zero events in the Poisson distribution, which is equal to e^{-1}, and there are b specific $2a$ edges that need to be added. Since the maximum size of a consequent cycle is 3, removing the $3a$ edge happens with the constant probability $1/3$. Hence, the probability of swapping a $2a$ edge with the $3a$ edge in a required triangle is $b/(3em)$ that happens in expected time $3em/b$ by the waiting-time argument. Let $T_{(1+1)\text{ EA-UM}}$ denote the first hitting time that (1+1) EA-UM finds the MST. Since b is at most n, we obtain the following upper bound on the expected time with probability $1 - o(1)$

$$\begin{aligned} E[T_{(1+1)\text{ EA-UM}}] &\leq \sum_{k=1}^{n} 3e \cdot \frac{m}{k} \leq 3en^2 H_n \\ &\leq 3en^2(\log n + 1) = O(n^2 \log n). \end{aligned}$$

Now we prove the lowe bound. Similar to the argument in the proof of the coupon collector's theorem (see, e. g., [MR95]), the lower bound $3en^2(\log n + 1) - cn^2$ holds with the probability $1 - e^{-e^c}$, if (1+1) EA-UM only adds one edge in each iteration. Setting $c = \frac{\log n}{2}$, the lower bound for the expected time is $\Omega(n^2 \log n)$ with probability $1 - o(1)$. Let k-step refer to the iterations that k triangle edges are chosen for the mutation step and note that $k \leq 3n/4$. It is enough to bound the contribution of k-steps on b during $\alpha n^2 \log n$ iterations for a constant $\alpha > 0$. The probability of a k-step for a constant $k \geq 1$ is

$$p_k^{\text{UM}} = \frac{e^{-1}}{(k-1)!} \cdot \binom{3n/4}{k} \cdot \left(\frac{1}{m}\right)^k = \theta(n^k m^{-k}) = \theta(n^{-k}),$$

where the first term is the probability of $k-1$ events in the Poisson distribution with $\lambda = 1$. Note that (1+1) EA-UM always adds at least one edge and $p_0^{\mathrm{UM}} = 0$. Within $\Theta(n^2 \log n)$ iterations, the expected number of 2-steps is $O(\log n)$ and there are $o(1)$ k-steps with $k > 2$. Each 2-step reduces b by at most 2. On the other hand, in a random spanning tree, each triangle contains a bad edge with probability $2/3$. Thus, b is at least $n/8 = \Theta(n)$ with probability $1 - e^{-\Omega(n)}$, using a Chernoff bound with $\delta = 2/8$. Hence, with the probability $1 - o(1)$, the expected time for (1+1) EA-UM to find the MST is

$$\begin{aligned} E[T_{\text{(1+1) EA-UM}}] &= \Theta\left(\sum_{k=1}^{b-2\log n} \frac{m}{k}\right) = \Theta\left(\sum_{k=1}^{b} \frac{n^2}{k} - \sum_{k=b-2\log n}^{b} \frac{n^2}{k}\right) \\ &= \Theta\left(\sum_{k=1}^{b} \frac{n^2}{k} - \sum_{k=1}^{2\log n} \frac{n^2}{k}\right) = \sum_{k=1}^{\Theta(n)} \frac{n^2}{k} - \sum_{k=1}^{O(\log n)} \frac{n^2}{k} \\ &= \Theta(n^2 \log n) - O(n^2 \log\log n) = \Theta(n^2 \log n). \end{aligned}$$

□

Now, we consider the performance of (1+1) EA-BM on the graphs G_1 and G_2.

Lemma 5.3. *Using the biased mutation with probability $q_{\mathrm{b}}(e)$, the probability of selecting edge e with rank $r = O(n)$ is $\Theta(1/n)$.*

Proof. Considering the denominator of $q_{\mathrm{b}}(e)$, we have

$$\begin{aligned} \sum_{i=1}^{m} a^{i/2} &= \frac{\sqrt{a} - a^{(m+1)/2}}{1-\sqrt{a}} = \frac{(1-o(1))\cdot\sqrt{a}}{1-\sqrt{a}} \\ &= \frac{(1-o(1))\cdot\sqrt{n-1}}{\sqrt{n}-\sqrt{n-1}} = \Theta(n). \end{aligned}$$

Since $r = O(n)$, for the numerator we have $1 \geq (1-\frac{1}{n})^{r/2} \geq (1-\frac{1}{n})^{cn} \geq e^{-c'}$, where $c < c'$ are constants. We conclude that $q_{\mathrm{b}}(e) = \frac{1-o(1)}{2e^{c'}n} = \Theta(1/n)$. □

Lemma 5.3 proves that the edges of G^T in G_1 have higher probability to be chosen in (1+1) EA-BM than in (1+1) EA-UM. In the following theorem, we show the effect of this property on the performance of (1+1) EA-BM.

Theorem 5.4. *(1+1) EA-BM finds the MST of G_1 in $\Theta(n \log n)$ with probability $1 - o(1)$.*

Proof. The proof is analogous to the proof of Theorem 5.2. However, we use Lemma 5.3 to tighten the probability of selecting edges from G^T. Hence, the expected waiting for the beneficial event in which a bad edge is removed from the tail is $\Theta(n/b)$. Thus, we obtain an upper bound of $O(n \log n)$.

To prove the lower bound, similar to the proof of Theorem 5.2, we use the argument of coupon collector's theorem with a similar approach used in [DJW02]. However, it

must be noted that we argue on the minimum number of edge selections such that all the bad edges are chosen for the mutation at least once. According to Lemma 5.3, the probability of selecting an edge in G^T is at least $1/cn$ for a constant c. Moreover, we have the initial number of bad edges is at least $n/8$ after the random initialisation with probability $1 - o(1)$. Note that (1+1) EA-BM selects at least one edge in each iteration.

Therefore, $(1 - 1/cn)^t$ is the probability of no triangle edge is selected after t iterations. Consequently, the probability of flipping at least one triangle edge in t iterations is $1 - (1 - 1/cn)^t$ that implies $(1 - (1 - 1/cn)^t)^{n/8}$ is the probability of selecting all of the $n/8$ bad edges at least once. Hence, the probability that at least one bad edge has never been selected in t iterations is $1 - (1 - (1 - 1/cn)^t)^{n/8}$. Finally, the probability that (1+1) EA-BM does not attempt to remove at least one bad edge in $t = (n-1)\ln n$ steps is $1 - (1 - (1 - 1/cn)^{(n-1)\ln n})^{n/8} \geq 1 - e^{-1/8c}$.

Therefore, (1+1) EA-BM needs $\Omega(n \log n)$ iterations to find the MST with probability of $1 - e^{-1/8c} - o(1)$, which completes the proof. □

Although (1+1) EA-BM efficiently finds the MST of G_1, the next argument shows that, in graphs similar to G_2, finding the MST takes exponential time.

Lemma 5.5. *The probability of selecting an edge with rank $r = \Omega(n^2)$ is exponentially small.*

Proof. According to the proof of Lemma 5.3, it is enough to show that the enumerator of $p(r)$ is exponentially small when $r > cn^2$ for some constant c. To this aim, we have

$$\left(1 - \frac{1}{n}\right)^{\frac{r}{2}} \leq \left(1 - \frac{1}{n}\right)^{\frac{c}{2}n^2} \leq e^{-\frac{c}{2}n} = O(e^{-n}).$$

□

Theorem 5.6. *The expected time for (1+1) EA-BM to find the MST of G_2 is exponential.*

Proof. In G_2, edges of G^T have higher weights than the edges of G^C. Since there are $\Omega(n^2)$ edges in G^C, the rank of edges of G^T is $\Omega(n^2)$. Using the result of Lemma 5.5, the probability of selecting any of the edges of G^T is $O(e^{-n})$. Hence, the expected time to select each of these edges for the mutation step is $\Omega(e^n)$. This implies that, in expectation, (1+1) EA-BM needs exponential time to reduce the value of b by one; consequently, it needs exponential time to find the MST of G_2. □

Before we continue with a result on arbitrary graphs we make a short trip into another solution encoding. Let $\mathcal{A}$ refer to the (1+1) EA_{MST} that uses a bit-string representation of the edges instead of spanning trees. Consider the *lollipop graph* presented in Figure 5.2 which consists of a clique with $n/2$ vertices and a path of length $n/2$ connected to it. Let all the edges of the clique have lower weights than all

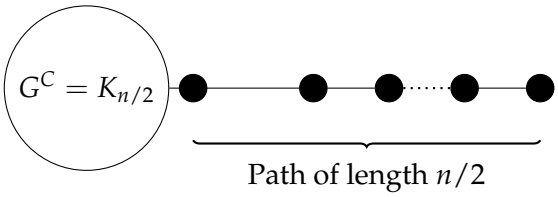


FIGURE 5.2: Worst case graph for random initialisation in the setting of bit-representation.

the edges of the path. Therefore, the rank of edges in the path is $\Omega(n^2)$ and have $q_b(e) = O(e^{-n})$. Creating a random sub-graph from the lollipop graph, the number of chosen edges from the tail is at most $2n/3$ with probability $1 - o(1)$. The lollipop graph illustrates that it is essential for $\mathcal{A}$ to be initialised with a spanning tree. Otherwise, it takes exponential time for it to find even a connected graph.

In the following, we analyse the effect of using both mutation strategies simultaneously in (1+1) EA-MM. Note that in every t iterations, (1+1) EA-MM performs $t/3$ uniform mutations and $t - t/3$ biased mutations with probability of $1 - o(1)$. This implies that repeating (1+1) EA-MM $c \geq 4$ times, the results of Theorems 5.2 and 5.4 also hold for the (1+1) EA-MM. However, since (1+1) EA-MM benefits from the uniform mutation in half of the iterations, it is also able to find the MST of G_2 in $\Theta(n^2 \log n)$.

This is the motivation to analyse the performance of (1+1) EA-MM on general graphs. For arbitrary graph G, let $w(T^i)$ be the weight of T^i, the spanning tree achieved by the algorithm in iteration i, and T^* be the minimum spanning tree. We define

$$g(T^i) = w(T^i) - w(T^*),$$

the weight gap that the algorithm needs to cover to reach the MST. Note that a MST is not necessarily unique but its weight is unique. We also redefine 1-step as an iteration that the algorithm adds only one edge and removes a random edge from the resulting cycle. Using a similar representation of Lemma 1 in [NW07], the following lemma presents how 1-steps contribute to reduce the value of $g(T)$.

Lemma 5.7. *Let solution T be an arbitrary spanning tree. There exists a set of $k \in \{1, \cdots, n-1\}$ different accepted 1-steps that if happen in any order transform T to T^* and reduce $w(T)$ by $g(T)/k$ on average.*

Proof. Let $E(T)$ and $E(T^*)$ denote the edges of T and T^*, respectively. Using an existence proof, Kano [Kan87] proved that there is a bijection $\alpha : E(T^*) \setminus E(T) \to E(T) \setminus E(T^*)$ such that $w(e) \leq w(\alpha(e))$ and adding e to T creates a cycle that includes $\alpha(e)$. Let $k = |E(T^*) \setminus E(T)|$. Swapping all the edges $e \in E(T^*) \setminus E(T)$ with α transforms T to T^* and reduces $g(T)$ to zero. Thus, each of these good swaps decreases the value of $g(T)$ on average by $g(T)/k$. Moreover, any 1-step that does a good swap is accepted since it results in a solution that is not worse than T. □

Using the result of Lemma 5.7 we prove a performance bound on (1+1) EA-MM on arbitrary graphs.

Theorem 5.8. *Starting from a random spanning tree, (1+1) EA-MM finds the minimum spanning tree in expected time* $O(n^3 \log(n \cdot w_{\max}))$*, where* $w_{\max}$ *is the maximum weight of the edges.*

Proof. Let $\Delta(g) = g(T^i) - g(T^{i+1})$ be the contribution of the algorithm in reducing the value of g in one iteration. The probability of having a 1-step equals to the probability of having zero events in the Poisson distribution which is $1/e$. Thus, with the probability of $1/(2enm)$, the uniform strategy causes a 1-step such that a specific edge e is added and a specific edge from the created cycle is removed. From Lemma 5.7 there are k good swaps. Therefore, the probability of a good swap in a 1-step with uniform strategy is $k/(2enm)$. Since a good swap reduces the value of $g(T)$ on average by $g(T)/k$, for $\Delta(g)$ we have

$$E[\Delta(g)] = \frac{g(T)}{k} \cdot \frac{k}{2enm}.$$

Since the the maximum value of $g(T)$ is $n \cdot w_{\max}$, using the multiplicative drift theorem [DG13] with $\delta = 1/(2enm)$, the expected first hitting time that $g(T) = 0$ is upper bounded by

$$\frac{\ln(n \cdot w_{\max}) + 1}{\left(\frac{1}{2enm}\right)} = O(n^3 \log(n \cdot w_{\max})).$$

□

Although (1+1) EA-MM guarantees a polynomial expected time to find T^* for any arbitrary graph, experiments by Raidl et al. showed that in many random graphs, all the edges of T^* have rank $O(n)$. This implies that the expected time for (1+1) EA-MM to find the MST improves to $O(n^2 \log(n \cdot w_{\max}))$ in many applications, since the probability performing a beneficial step improves to $1/(2em)$.

5.4 Multi-Objective Problem

In this section we consider the multi-objective version of the minimum spanning tree problem. Firstly, we introduce the ranking of the edges in multi-objective space and experimentally show a considerably good approximation for the appearance of edges in an moMST according to their ranks. Using the approximation, we analyse the performance of G-SEMO-UM and G-SEMO-BM dealing with two different types of graphs.

5.4.1 Experimental Approximation

The work by Raidl. et al. [RKJ06] considered the single-objective scenario and lays the groundwork for our empirical study. As a reminder: the authors showed that low rank edges have a much higher probability to be part of MSTs. In the setting of multiple conflicting objectives similar assumptions are reasonable, i. e., that non-dominated spanning trees are more likely composed of "low-rank" edges for an appropriate definition of "rank". In a recent study Bossek et al. [BGN19] considered different ranking definitions in the bi-objective case. More precisely, they considered (1) the non-domination level and (2) the domination number of an edge to define the rank and established a total order on the edges with low ranks being favoured. Similar to Raidl's work, they conducted an empirical study and estimated the probability $p^{\mathrm{m}}(r)$ of edges to be part of at least one spanning tree as a function of its rank r for different graph classes (more details in the following). They, next, empirically evaluated the convergence speed of biased edge selection strategies in comparison to the baseline of random uniform selection. They obtained significant improvements, particularly in the case where the domination number $d(e) = |\{e' \in E \,|\, w(e') \succeq w(e)\}|$ was adopted for the definition of rank and the probability of choosing an edge with edge r for insertion was set to

$$q_{\mathrm{b}}^{\mathrm{m}}(r) = \frac{p^{\mathrm{m}}(r)}{\sum_r p^{\mathrm{m}}(r)},$$

i. e., proportional to its probability of appearance in non-dominated solutions. We catch up on their work and illustrate empirically, that $\beta \cdot ((n-1)/n)^r$ – similar to Raidl's results – is indeed a good approximation for the probability $p^{\mathrm{m}}(r)$. In line with Bossek et al., our empirical study is based on different graph types reflecting different levels of density and edge weight distribution. Complete graphs (CEG for Complete Edge Generation) with n nodes placed uniformly at random in $[0, 100]^2$ are studied alongside graphs where the interconnection of nodes is based on a Delauney triangulation of the point cloud in the Euclidean plane (Delauney Edge Generation). Note that in the latter case $m = \Theta(n)$. Edge weights $w_i(e), i = 1, 2$ either both are realisations of uniform random numbers stemming from a $\mathcal{U}[5, 200]$-distribution (RNDRND; in consistence with [KC01; ZG99]) or the first weight corresponds to the Euclidean distance between the nodes in the plane and the second weight is sampled from a $\mathcal{U}[5, 200]$ distribution (EUCRND). For each graph type, i. e., CEG-RNDRND, CEG-EUCRND, DEG-RNDRND and DEG-EUCRND we consider $n \in \{25, 50, 100, 250\}$.

The estimation of $p^{\mathrm{m}}(r)$ follows [BGN19]. Here, we describe the procedure in a nutshell and refer the interested reader to the original work. First consider a single random graph $G = (V, E)$ of a given graph type and problem size n. For each

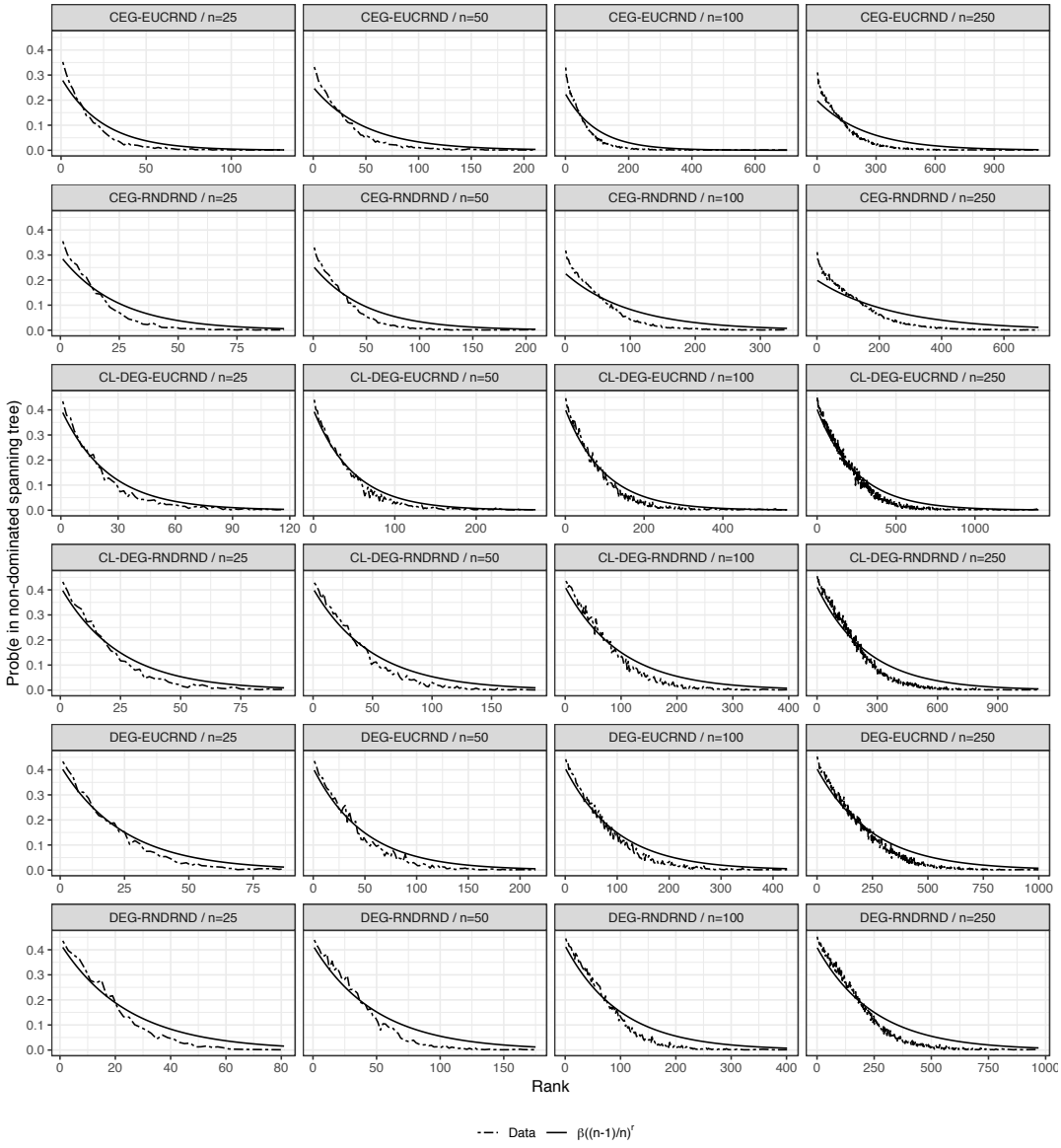


FIGURE 5.3: Empirical probabilities $p^{\mathrm{m}}(r)$ of edges to be part of at least one non-dominated spanning tree as a function of its rank r measured by the domination number (lower is better). The empirical data is accompanied by regression models of the form $\beta \cdot ((n-1)/n)^r$. See Table 5.1 for supportive results of a regression analysis.

edge e, we calculate the number of non-dominated spanning trees that e is part of [2], termed the share $s(e)$, and estimate the probability of r-ranked edges by the average of all shares of the corresponding rank. We repeat this process for 1000 random graphs of the corresponding graph type and $n \in \{25, 50, 100, 250\}$ and use the mean probability over all 1000 instances as the final estimate for $p^{\mathrm{m}}(r)$.

Figure 5.3 shows the estimations of $p^{\mathrm{m}}(r)$, the probability of rank-r edges to be part

[2]The set of non-dominated spanning trees is approximated by a simple weighted-sum approach minimising $\lambda w_1(T) + (1-\lambda)w_2(T)$ for equidistantly sampled $\lambda = k/1000, k = 0, \ldots, 1000$.

TABLE 5.1: Results of regression analysis separated by graph class and instance size.

Graph class	n	R^2	**RMSE**	β
CEG-EUCRND	25	0.9434	0.0243	0.2900
	50	0.9240	0.0263	0.2515
	100	0.9113	0.0264	0.2253
	250	0.8893	0.0289	0.1988
CEG-RNDRND	25	0.9315	0.0299	0.2960
	50	0.9196	0.0299	0.2559
	100	0.9062	0.0297	0.2270
	250	0.8954	0.0298	0.1996
DEG-EUCRND	25	0.9802	0.0184	0.4054
	50	0.9832	0.0166	0.4006
	100	0.9784	0.0198	0.4024
	250	0.9712	0.0269	0.4041
DEG-RNDRND	25	0.9647	0.0336	0.4269
	50	0.9566	0.0351	0.4174
	100	0.9570	0.0347	0.4152
	250	0.9540	0.0360	0.4095

of at least one non-dominated spanning tree, separated by graph class and number of nodes. The estimations are accompanied by fitted regression models of the form $\beta \cdot ((n-1)/n)^r$. We observe that the model mostly adheres quite well to the data. These apparent observations a supported by the results of a regression analysis shown in Table 5.1. Here, the R^2 values – a measure for the fraction of variance in the data explained by the model – takes values close to 1 with a minimum of 0.8893 for CEG-EUCRND graphs with $n = 250$ nodes. Additionally, the root mean squared error (RMSE) values, i. e., the mean deviation of the model predictions to the data, are very low consistently. All in all the experiments support our parametric model assumption for different dense and sparse graphs. As a consequence, we use this empirical estimate for our upcoming theoretical runtime analysis.

5.4.2 Theoretical Analysis

Motivated by the experimental results, we use $p^{\mathrm{m}}(r) = \beta \cdot ((n-1)/n)^r$ as the approximation for the probability of an edge with domination number r appears in the moMST. As β consistently takes values in $(0, 1)$ throughout the experiments, we drop this constant factor in subsequent investigations. Note that we break rank ties randomly. Hence, we have $m = |E|$ different edges with m different probabilities. Using Bossek et al. [BGN19] approach, for each edge e with domination number r we set

$$q(e) = q_{\mathrm{b}}^{\mathrm{m}}(r)$$

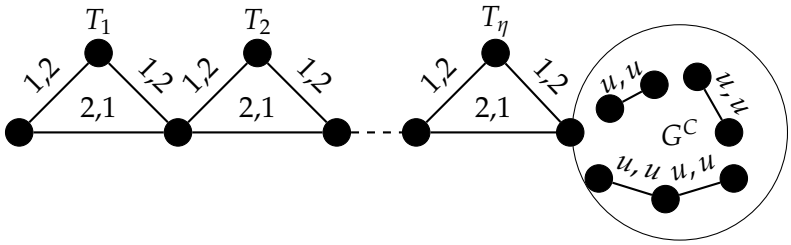


FIGURE 5.4: Triangular-tailed graph G with a chain of $p = n/4$ triangles and a giant component $G^C = K_{n/2}$. [NW07]

for the probability of choosing e in the mutation step in Algorithm 12. Using the same arguments as in Lemma 5.3, we have the following lemma for biased mutation in the multi-objective setting.

Lemma 5.9. *Using the biased mutation with probability $q(e) = q_b^m(r)$, the probability of selecting edge e with domination count $r = O(n)$ is $\Theta(1/n)$.*

Again, we consider the triangular-tailed graph in two versions G_1^m and G_2^m. Both graphs contain η triangles in the tail. In each triangle, the two upper edges have weights $(1,2)$ while the bottom edge has weight $(2,1)$. The difference lies in the composition of the clique part G^C. Here, in G_1^m all edges have the same weight (k,k), $k > 2$ while in G_2^m there exists a subset $G^S = \{e_1, \ldots, e_l\} \subseteq G^C$ of size $l \leq (n/2 - 1)$ with $w(e) = (u,u)$, $u > 2$, for each edge $e \in G^S$ and $w(e) = (k,k)$, $k > u + n + 1$, for all remaining clique edges. We also assume that the edges in G^S do not create any cycle. Let us at this point retain the following: every non-dominated spanning tree of G_1^m contains an arbitrary spanning tree on G^C as a sub-graph. In contrast, in G_2^m every non-dominated spanning tree must necessarily contain G^S as a sub-graph.

Let us briefly state our goals here. We denote by $\mathcal{T}^*$ the set of non-dominated spanning trees for a given graph and by $\mathcal{F} = w(\mathcal{T}^*)$ its image, i. e., the set of all Pareto-optimal objective vectors. We seek to locate for each $f \in \mathcal{F}$ a spanning tree $T^* \in \mathcal{T}^*$.

Lemma 5.10. *For both G_1^m and G_2^m we have $|\mathcal{F}| = \Theta(n)$.*

Proof. Let us first consider the clique part. In G_1^m each spanning tree of G^C has equal weight, we may fix an arbitrary one. In G_2^m the non-dominated spanning tree of G^C must include all the edges of G^S. Thus, for each graph type G_1^m and G_2^m, the contribution of the edges of G^C in objective values are the same. Since the triangular tail is identical for both G_1^m and G_2^m, the following observations hold for both versions. Every non-dominated spanning tree contain exactly two edges of each triangle, in particular at least one edge with weight $(1,2)$, i. e. there are at least η edges of weight $(1,2)$ in each Pareto solution. Hence, for each non-dominated spanning tree the weight of the triangular part is

$$\eta \cdot \begin{bmatrix} 1 \\ 2 \end{bmatrix} + r \cdot \begin{bmatrix} 1 \\ 2 \end{bmatrix} + (\eta - r) \cdot \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 3\eta - r \\ 3\eta + r \end{bmatrix},$$

where $0 \leq r \leq \eta$ is the number of triangles that have two upper edges in the spanning tree. Together with our observations in the clique part, this implies $r \in \Theta(n)$ and, as a direct consequence, $|\mathcal{F}| = \Theta(n)$. □

Let $f_0, f_1, \ldots, f_\eta \in \mathcal{F}$ be the objective vectors in ascending order of the first weight (and thus in descending order of the second weight). In the following, we show that we can easily move between Pareto-optimal spanning trees with distinct weights. We use the notation $d(T, T') := |T \setminus T'|$ and speak about distance of spanning trees in terms of the necessary edge exchange operations needed to transform T to T'.

Lemma 5.11. *For each non-dominated spanning tree T in G_1^m and G_2^m with $w(T) = f_i, 0 \leq i \leq \eta$, there is a non-dominated spanning tree T' with $d(T, T') = 1$ such that*

- $w(T') = f_{i+1}$ *for* $0 \leq i \leq \eta - 1$ *or*
- $w(T') = f_{i-1}$ *for* $1 \leq i \leq \eta$.

Proof. We only prove the first case. The proof for the other case is similar. Consider a non dominated spanning tree T with $w(T) = f_i, 0 \leq i \leq \eta - 1$. T contains exactly $(\eta - i)$ edges of weights $(2, 1)$ in the triangular-tail part. Now we obtain T' by including one of the $\eta - (\eta - i) = i$ remaining edges of weight $(2, 1)$ and dropping a $(1, 2)$ weighted edge on the resulting cycle. It follows that $w(T') = f_{i+1}$ and clearly $d(T, T') = 1$. □

Lemma 5.11 states that once we found a single non-dominated spanning tree it is easy to obtain the others.

Theorem 5.12. *On G_1^m, given an initial spanning tree T, G-SEMO-UM needs expected time $O(n^3 \log n)$ to cover the Pareto front.*

Proof. Let T be a spanning tree with $w(T) = f_i$ and $b(T)$ denote the number of triangle edges with weight $(2, 1)$ for T. We shall refer those edges *bottom edges* in the following. Since, $w(T) = f_i$ clearly $b(T) = i$. By Lemma 5.11 we can move to a tree with weight vector f_{i+1} or f_{i-1} by adding or removing a bottom edge. In G-SEMO$_{\text{MST}}$ (see Algorithm 12) achieving $f(i+1)$ happens with probability at least

$$\left(\frac{1}{i+1}\right) \cdot \left(e^{-1} \cdot \frac{(\eta - i)}{m}\right) \cdot \left(\frac{2}{3}\right) = \frac{2(\eta - i)}{3em(i+1)}.$$

Here, the first term is the probability to select the individual T with $w(T) = f_i$ such that T' with $w(T') = f_{i+1}$ is not included in the population yet, the second term is the probability for the 0 event of a Pois$(\lambda = 1)$ distribution, i. e., to add exactly one edge to the sampled solution, and the third term is the probability to remove one of the non-bottom edges from the resulting cycle. Adopting waiting time arguments, the expected number of iterations until f_{i+1} is achieved is bounded from above by

$3em(i+1)/2(\eta-i)$. Hence, the total time until the population of G-SEMO$_{\text{MST}}$ contains each one solution for each Pareto-optimal objective vector $f_i, i = 0, \ldots, \eta$ – only by adding bottom edges and starting with a solution with trade-off f_0 in the worst case – is bounded by the sum

$$\sum_{i=0}^{\eta-1} \frac{3em(i+1)}{2(\eta-i)} = \frac{3em}{2} \cdot \sum_{i=0}^{\eta-1} \frac{(i+1)}{(\eta-i)} \leq \frac{3em}{2} \cdot \sum_{i=0}^{\eta-1} \frac{\eta}{(\eta-i)}$$
$$= \frac{3em\eta}{2} \cdot H_\eta = O(n^3 \log n).$$

On the other hand, to include f_{i-1}, the algorithm must choose a non-bottom edge from the triangles that include one, with probability $i/(em)$, and remove the bottom edge with probability $1/3$. Thus, the probability of this event is $i/3em(i+1)$, i. e., the expected number of iterations for such event happen is $3em(i+1)/i$. Therefore – only by decreasing the number of bottom edges and starting from f_η in the worst case – the expected time for G-SEMO$_{\text{MST}}$ to achieve all the objective vectors in the Pareto front is upper bounded by the sum

$$\sum_{i=1}^{\eta} \frac{3em(i+1)}{i} = 3em \cdot \sum_{i=1}^{\eta} \frac{(i+1)}{i} \leq 3em \cdot \sum_{i=1}^{\eta} \frac{\eta}{i}$$
$$= 3em\eta \cdot H_\eta = O(n^3 \log n).$$

All together, since one of the cases is always available, the total upper bound is $O(n^3 \log n)$. □

Next we consider the performance of G-SEMO-UM on G_2^{m}. In an arbitrary moMST T of G_2^{m}, let $s = |G^S \cap T|$ denote the number of optimal edges G^S in T.

Lemma 5.13. *For two solutions $T_1, T_2 \in G_2^{\text{m}}$, $T_1 \succ T_2$ if and only if $s_1 > s_2$.*

Proof. Considering the proof of lemma 5.10, the difference between the objective values of T_1 and T_2 that is caused because of the chosen tail edges is at most n. On the other hand, increasing s improves both objective values by at least $n+1$. Thus any solution that has larger s have strictly better objective value in both objectives. □

Lemma 5.13 shows that all the solutions in the population set of G-SEMO$_{\text{MST}}$ have the same value of s. Note that G-SEMO$_{\text{MST}}$ starts with a spanning tree and any offspring is also a spanning tree.

Theorem 5.14. *On G_2^{m}, given an initial spanning tree T, G-SEMO-UM needs expected time $O(n^3 \log n)$ to cover the Pareto-front.*

Proof. We consider two phases in this proof. The first phase is to find the solution T with $s = l$, i. e. T contains all the edges of G^S. After this phase, we know that any offspring is a Pareto-optimal solution. The next phase is to cover the whole Pareto front.

Note that all the solutions in the current population have the same value of $s < l$. Thus, the probability of choosing solution T with highest s is 1. To increase s, the algorithm adds edge $e \in G^S \setminus T$ with probability $(l-s)/m$. Adding e can cause a cycle with size at most n. In the worst case, there is only one edge in the cycle that can be removed without removing another optimal edge. Hence, a beneficial removing happens with probability of $1/n$. Therefore, the probability of increasing s by 1 is at least $e^{-1} \cdot \frac{l-s}{m} \cdot \frac{1}{n}$, where e^{-1} is the probability that G-SEMO$_{\text{MST}}$ adds only one edge. Such mutation step happens after $O(mn/(l-s))$ iterations in expectation. The minimum initial value for s is zero and l is at most $n-1$. Thus, the expected time for G-SEMO$_{\text{MST}}$ to finish phase one is upper bounded by

$$\sum_{i=0}^{l-1} \frac{mn}{e(l-s)} \leq n^3 \sum_{i=1}^{n} \frac{1}{n} = O(n^3 \log n)$$

In the second phase, G-SEMO$_{\text{MST}}$ does not accept a solution with $s < l$. Hence, the same argument as in Theorem 5.12 proves that G-SEMO$_{\text{MST}}$ finishes the second phase in $O(n^3 \log n)$ expected time and this completes the proof. □

Now we consider G-SEMO-BM algorithm with biased mutation that select the edges with $q(e) = q_{\text{b}}^{\text{m}}(r)$. The number of edges in the tail of G_1^{m} and G_2^{m} is the same and equal to $3n/4$. In both of the graphs, these edges dominate every other edges and consequently have lower non-domination ranks, i. e. each edge have a unique random rank within $\{1, \cdots, 3n/4\}$. Moreover, in G_2^{m}, edges of G^S dominate other edges of G^C. Hence, ranks $3n/4, \cdots, (3n/4)+l$ belong to the edges of G^S. Therefore, as Lemma 5.9 shows, all the edges that belong to the moMSTs in G_1^{m} and G_2^{m} have the selection probability $\Theta(1/n)$. Using the same arguments as in Theorems 5.12 and 5.14, the following result hold for the performance of G-SEMO$_{\text{MST}}$ on the graphs G_1^{m} and G_2^{m}.

Corollary 5.15. *On G_1^{m} and G_2^{m}, given an initial spanning tree T, G-SEMO-BM needs expected time $O(n^2 \log n)$ to cover the Pareto-front.*

5.5 Conclusion

We performed a rigorous asymptotic runtime analysis of evolutionary algorithms with biased mutation for the classic Minimum Spanning Tree problem. Bias in this

context means that edges of low weight in the single-objective case and of low domination number in the multi-objective case are assigned a higher probability of mutation. Our findings reveal that bias is blessing and curse at the same time. While a significant time complexity speedup can be achieved in some cases, bias may also lead to exponential expected optimisation time if edges of high rank are part of optimal solutions. We showed that using the biased and unbiased mutations simultaneously is the key to avoid the extreme cases of bias.

Chapter 6

Baseline Evolutionary Algorithms for the Packing While Travelling Problem

6.1 Introduction

While a number of simple nonlinear problems, such as trap functions and LeadingOnes, have been deeply studied from theoretical and practical aspects [AD18; GK16; NB03], the literature is not equally rich for investigations of EAs on non-linear functions. Specifically, multi-component problems have gained a lot of attention during recent years due to their appearance in many real-world applications [Bon+19]. Multi-component problems consist of different components integrating with each other in a way that it is essential to deal with all the components simultaneously to find good quality solutions.

In this chapter, we theoretically analyse the performance of three baseline evolutionary algorithms, RLS_swap, (1+1) EA and G-SEMO$_e$, on variations of the packing while travelling problem, which has been introduced in Section 3.4 and prove upper bounds for the expected running times. RLS_swap and G-SEMO$_e$ are modified versions of RLS and G-SEMO algorithms that are introduced in Sections 2.2.1 and 2.3.7, respectively. We prove that for the instances with correlated weights and profits, RLS_swap and G-SEMO$_e$ find the optimal solution in expected time $O(n^3)$. Furthermore, we consider the instances with uniform weights and prove that (1+1) EA finds the optimal solution in expected time $O(n^2 \log(\max\{n, p_{\max}\})$. We also investigate the performance of these algorithms in addition to two other multi-objective algorithms, which are introduced in section 6.4, from the experimental point of view.

The work of this chapter is based on a conference paper presented at 15th workshop on foundations of genetic algorithms (FOGA XV) [RPN19]. The rest of the chapter is organised as follows. Section 6.2 presents the detailed definition of considered PWT problem and the algorithms used in this study. In Section 6.3, we investigate the problem theoretically for correlated weights and uniform weights in Sections 6.3.1

and 6.3.2, respectively. Our experimental analyses are presented in Section 6.4, followed by a conclusion in Section 6.5.

6.2 Preliminaries

In this section we present the version of PWT problem that is considered in this chapter as well as the details of algorithms that we analyse.

6.2.1 Problem Definition

In this chapter, we consider a version of PWT which is slightly different from the one introduced in Section 3.4. In this version, there are only two cities and n items that are located in the first city. In addition, the weights of the given items are favourably correlated with the profits, i.e. for any two item e_i and e_j, $p_i > p_j$ implies $w_i < w_j$. Hence, for a bit string solution $s = (x_1 \cdots x_n) \in \{0,1\}^n$, we have

$$B(s) = \sum_{i=1}^{n} x_i \cdot p_i - \frac{Rd}{v_{\max} - \nu \sum_{i=1}^{n} x_i \cdot w_i},$$

where d is the distance between the two cities.

We assume there are no items with the exact same weight and profit, however, all the proofs can be extended to include these cases and achieve the same results. Moreover, without loss of generality, in the rest of the chapter we assume that items are indexed in a way that $p_1 \geq p_2 \cdots \geq p_n$ and $w_1 \leq w_2 \cdots \leq w_n$. We also consider another version of the problem in which all the weights are uniform and equal to one.

6.2.2 Algorithms

In this chapter we study the behaviour of three algorithms. The first one, described in Algorithm 13, is a RLS variant called RLS_swap, which is able to do a swap (flip a zero bit and a one bit simultaneously), in addition to the usual one-bit flip. This modification of the classical RLS has been previously considered for the MST problem in [NW07]. In each iteration, if the current solution is all zeros or ones, it flips a randomly chosen bit of the solution. Otherwise, with probability of $1/2$, it either does a one-bit flip as described or chooses a one and a zero uniformly at random and flips both of them. The generated offspring replaces the current solution if it is at least as good as its parent with respect to the fitness function. This swap mutation is added in RLS_swap because there are some situations in optimising PWT in which no one-bit flip is able to pass the local optima.

Another algorithm we consider is (1+1) EA (Algorithm 3). This algorithm, as introduced in Section 2.3.6 flips each bit of the current solution with probability of $1/n$ in

Algorithm 13: RLS_swap

```
Choose s ∈ {0,1}^n uniformly at random;
Let |s|_1 denote the number of items in s;
while stopping criterion not met do
    p ← a random real number in [0,1];
    if |s|_1 = 0 ∨ |s|_1 = n ∨ p < 1/2 then
        Create s' by flipping a randomly chosen bit of s ;
    else
        Create s' by flipping a randomly chosen zero bit and a randomly chosen
          one bit of s ;
    if F(s') ≥ F(s) then
        s ← s';
```

each mutation step. Similar to RLS_swap, it compares the parent and the offspring and picks the better one for the next generation.

In RLS_swap and (1+1) EA, which are single-objective algorithms, the comparisons between solutions is based on the fitness function

$$F(s) = (q(s), B(s))$$

where $q(s) = \min\{C - w(s), 0\}$. According to $q(s)$, s is infeasible if and only if $q(s) < 0$ and the absolute value of $q(s)$ denotes the amount of constraint violation. The goal is to maximise $F(s)$ with respect to lexicographic order, i.e. s_1 is better than s_2 ($F(s_1) \geq F(s_2)$) if and only if $(q(s_1) > q(s_2)) \vee (q(s_1) = q(s_2) \wedge B(s_1) \geq B(s_2))$. This implies that any feasible solution has better fitness than any infeasible solution. Moreover, between two infeasible solutions, the one with smaller constraint violation is better.

We also consider PWT with a multi-objective algorithm using a variant of G-SEMO algorithm introduced in Section 2.3.7, which uses a specific selection function to deal with the exponential size of the population (Algorithm 9). Neumann and Sutton suggested G-SEMO$_e$ for the knapsack problem with correlated weights and profits to avoid the effects of an exponential population size [NS18]. We use the same approach since PWT easily changes to KP when $R = 0$. As the objectives, we use the weight function ($W(s)$) and the previously defined fitness function ($F(s)$). The aim is to minimise $W(s)$ while maximising $F(s)$. Between two solutions s_1 and s_2, we say s_1 (weakly) dominates s_2, denoted by $s_1 \succeq s_2$, if and only if $W(s_1) \leq W(s_2) \wedge F(s_1) \geq F(s_2)$. The dominance is called strong, denoted by $s_1 \succ s_2$, when at least one of the inequalities strictly holds. Note that based on this definition, similar to the single-objective fitness function, each feasible solution dominates all infeasible solutions and an infeasible solution closer to the constraint bound dominates the more distant ones.

Algorithm 14: G-SEMO$_e$

Choose $s \in \{0,1\}^n$ uniformly at random;
$P \leftarrow \{s\}$;
while *stopping criterion not met* **do**
  Let $P_i = \{s \in P \mid |s|_1 = i\}, 0 \leq i \leq n$ and $I = \{i \mid P_i \neq \emptyset\}$;
  Choose $j \in I$ uniformly at random;
  $s \leftarrow \arg\max\{B(x) \mid x \in P_j\}$;
  Create s' by flipping each bit of s independently with probability of $1/n$;
  if $\{z \in P : z \succ s'\} = \emptyset$ **then**
    $P \leftarrow P \setminus \{z \in P \mid s' \succeq z\} \cup \{s'\}$;

6.3 Theoretical Analysis

In this section, we theoretically investigate the performance of RLS_swap, G-SEMO$_e$ and (1+1) EA on different versions of the PWT problem by using the running time analysis. We first consider RLS_swap and G-SEMO$_e$ on the PWT with correlated weights and profits. Next, the behaviour of (1+1) EA on the PWT problem with uniform weights is analysed.

To study the PWT problem, we need to investigate the properties of an optimal solution and the impact of adding or removing an item on the benefit function. For this reason, in the following lemma we prove that the weight of the current solution, w_i, and p_i determine if item e_i is worth adding to or removing from the current solution.

Lemma 6.1. *For each item e_i there is a unique threshold w_{e_i} such that adding e_i to the current solution s improves the benefit function if and only if $W(s) < w_{e_i}$. Moreover, $W(s) > w_{e_i} + w_i$ if and only if removing e_i increases the benefit function.*

Proof. Assume that the current solution s does not include item e_i and $s' = s \cup e_i$. Hence, we have

$$\begin{aligned} B(s') - B(s) &= (P(s') - RT(s')) - (P(s) - RT(s)) \\ &= p_i - Rd\left(\frac{1}{v_{\max} - \nu(W(s) + w_i)} - \frac{1}{v_{\max} - \nu W(s)}\right) \\ &= p_i - \frac{Rd\nu w_i}{(v_{\max} - \nu(W(s) + w_i)) \cdot (v_{\max} - \nu W(s))}. \end{aligned} \tag{6.1}$$

Adding e_i to s increases $B(s)$ only if the value of Expression 6.1 is greater than zero. Solving this equation we have

$$B(s') - B(s) \geq 0 \iff W(s) \leq \overbrace{\frac{v_{\max}}{\nu} - \frac{w_i}{2}\left(1 + \sqrt{1 + \frac{4Rd}{\nu w_i p_i}}\right)}^{w_{e_i}}. \tag{6.2}$$

By solving Equation 6.1 for each e_i, $1 \leq i \leq n$, we can find w_{e_i} such that adding e_i to s improves $B(s)$ if and only if $W(s) < w_{e_i}$. Considering the case that $e_i \in s$ and $s' = s \setminus \{e_i\}$, a similar calculation implies that

$$B(s') - B(s) \geq 0 \iff W(s) \geq \frac{v_{\max}}{\nu} + \frac{w_i}{2}\left(1 - \sqrt{1 + \frac{4Rd}{\nu w_i p_i}}\right) \geq w_{e_i} + w_i. \tag{6.3}$$

In other words, removing e_i from s increases the benefit function if and only if $W(s) > w_{e_i} + w_i$, which completes the proof. □

A direct result from Equations 6.2 and 6.3 is that

$$p_i \geq p_j \wedge w_i \leq w_j \implies w_{e_i} \geq w_{e_j}.$$

Therefore, we have $w_{e_1} > w_{e_2} > \cdots > w_{e_n}$. Moreover, if the weight of solution s equals w_{e_i} then $B(s) = B(s \cup e_i)$. Note that for any item e_i and e_j, $i < j$, if removing e_i is beneficial, then it is the same for e_j because of the correlated weights and profits. Hence, for any $1 \leq i < j \leq n$, we also have

$$w_{e_i} + w_i > w_{e_j} + w_j. \tag{6.4}$$

Now we discuss the optimal solution of the PWT problem. Let $s_i = (1^i, 0^{n-i})$ denote the solution that only includes the first i items and $s_0 = (0^n)$. Consider the case that for some i, we have $w_{e_i} > W(s_{i-1})$. Hence, by Lemma 6.1, adding e_i to s_{i-1} improves the benefit function and $B(s_i) > B(s_{i-1})$. Moreover, since $w_{e_1} \geq \cdots \geq w_{e_i}$, we have $B(s_i) \geq \cdots \geq B(s_1)$. The claim is that for a given set of items, there is a unique k such that the optimal solution of the PWT problem is packing either k or $k \wedge (k+1)$ items with the lowest indices. In the second case, s_k and s_{k+1} are both optimal and have the same benefit value. The following lemma proves this claim.

Lemma 6.2. *The optimal solution for the Packing While Travelling problem is the set of k or $k \wedge (k+1)$ items with highest profits and lowest weights where k is unique and depends on the given set of items.*

Proof. First, we assume that $w_{e_n} > W(s_{n-1})$. In this case, we have $B(s_n) > B(s_{n-1})$ and s_n is the optimal solution. On the other hand, if $w_{e_1} < 0$ then the optimal solution is s_0. In the rest of the proof, we assume that neither of these cases happen.

Let $o = \min\{i \mid W(s_i) > w_{e_{(i+1)}}, 0 \leq i \leq n-1\}$. According to the definition of o and $w_{e_{o+1}}$, adding any item to s_o decreases the benefit function. Moreover, we have $W(s_{o-1}) < w_{e_o}$, which implies $W(s_{o-1}) + w_o = W(s_o) < w_{e_o} + w_o$. Hence, removing any item from s_o also reduces the benefit function. Therefore, the following equation

holds:

$$B(s_0) < \cdots < B(s_o) \geq B(s_{o+1}) > \cdots > B(s_n).$$

The equality $B(s_o) = B(s_{o+1})$ holds only when $w_{e_{o+1}} = W(s_o)$. However, all other inequalities are strict according to Lemma 6.1.

To prove that s_o or $s_o \wedge s_{o+1}$ are the only optimums, it is now enough to show that for any i, s_i has the highest benefit value among other solutions with i items. This is also true since the items in s_i have the highest profits and the lowest weights, which result in the highest benefit value.

Involving the capacity constraint C, however, may change the optimal solution. We define s_k, $k = \max\{j \mid W(s_j) \leq C, 0 \leq j \leq o\}$, the feasible solution with the highest benefit function, which is the actual optimal solution with respect to C. This finalises the proof.

□

From this point, we denote the optimal solution by s^* and $k = |s^*|$ denotes the number of selected items in the optimal solution.

6.3.1 Correlated Weights and Profits

In this section, we consider the instances of the PWT problem in which the weights are strongly correlated with the profits. We calculate the performance of RLS_swap and G-SEMO$_e$ for this type of PWT.

RLS_swap

Using the result of Lemma 6.2, we analyse the performance of RLS_swap finding the optimal solution of the PWT problem in terms of the number of evaluations. We refer to the first k bits of s^* as the first block and the rest as the second block. Let l and r denote the number of zeros in the first block and the number of ones in the second block, respectively. For technical reason, we assume item e_0 exists where $w_{e_0} > \sum_{i=1}^{n} w_i$ and $x_0 = 1$. We denote the solution achieved by RLS_swap after t generations as s^t. Consequently, we define

$$h_t = \max\{0 \leq i \leq k \mid x_0 = x_1 = \cdots = x_i = 1 \wedge W(s^t) < w_{e_i} + w_i\},$$

to be the index of a specific bit of s^t. The following lemma and theorem consider the performance of RLS_swap on the PWT problem with correlated weights.

Lemma 6.3. *Having obtained a solution s^t, RLS_swap does not accept a solution $s' =< x'_1, \cdots, x'_n >$ in which $\exists i \leq h_t : x'_i = 0$.*

Proof. Since the weights and profits are correlated, it is enough to prove RLS_swap does not remove e_{h_t}. Let s' denote the solution in generation $t' > t$ such that $W(s') \geq$

$w_{e_{h_t}} + w_{h_t}$ for the first time after t, i.e. RLS_swap is able to remove e_{h_t} from s'. Note that all the items e_i, $i \leq h_t$, are still in s' at time t'. Furthermore, no swap mutation can remove e_{h_t} since it has a higher profit value and less weight than other missed items. Hence, there exists an item e_x, $x > h_t$, that has been added to s' with a one-bit flip in iteration $t' - 1$ such that $W(s') - w_x < w_{e_{h_t}} + w_{h_t}$, $C \geq W(s') \geq w_{e_{h_t}} + w_{h_t}$ and $B(s') \geq B(s' \setminus e_x)$. From the benefit inequality and Lemma 6.1, we have

$$W(s') - w_x \leq w_{e_x} \Rightarrow W(s') \leq w_{e_x} + w_x.$$

Since $x > h_t$, according to Inequality 6.4, we have $w_{e_{h_t}} + w_{h_t} > w_{e_x} + w_x$. Hence, $W(s') < w_{e_{h_t}} + w_{h_t}$ which contradicts with the assumption that it is beneficial to remove e_{h_t} from s' and completes the proof. □

According to the definition of h_t, the weight of the accepted solutions after generation t is lower bounded by $W(s_{h_t})$, in which the first h_t items are selected. This shows that the maximum possible value for h_t is k, otherwise s^* is not the optimal solution (Lemma 6.2). Let

$$y = \max\{h_t, \min\{i \mid h_t < i \leq n \wedge x_i = 1\}\}, \tag{6.5}$$

be the index of the first one bit after a sequence of zeros after h_t. Let $y = h_t$ if there is no one bit after h_t. Similarly to the proof of Lemma 6.3, the following corollary holds.

Corollary 6.4. *Obtaining a solution s with $W(s) < w_{e_y} + w_y$, RLS_swap does not remove e_y except by reducing the value of y.*

Reducing the value of y to $y' < y$ is either caused by swapping e_y with item $e_{y'}$ or inserting $e_{y'}$ with a one-bit flip. In the first case, due to Inequality 6.4, and in the second case, due to Lemma 6.1, we have $W(s') < w_{e_{y'}} + w_{y'}$, i. e. Corollary 6.4 holds for y'.

Theorem 6.5. *RLS_swap finds the optimal solution for the Packing While Travelling problem with correlated weights and profits in $O(n^3)$ expected time.*

Proof. Let RLS_swap start with a random solution s^0. We analyse the optimisation process in three main phases. In the first phase, RLS_swap finds a feasible solution. The second phase is to obtain $h_t = k$ and the third phase is to remove the remaining items from the second block to achieve the optimal solution.

If $W(s^0) \leq C$ the first phase is already complete. Therefore, let $W(s^0) > C$. Using a fitness level argument, Neumann and Sutton proved that (1+1) EA, which uses a fitness function with strictly higher priority in weight constraint satisfaction, finds a feasible solution for KP with correlated weights and profits in $O(n^2)$ expected time (Theorem 3 in [NS18]). Their proof also holds for RLS_swap since the constraint is linear and RLS_swap is able to do a one-bit flip in $O(n)$ expected time. Hence,

RLS_swap finds a feasible solution s in $O(n^2)$ expected time and completes the first phase.

In the second phase, we analyse the expected time needed to increase the value of h_t to $h_t = k$. To do this, we need to calculate the expected time to find a solution s such that $x_{h_t+1} = 1$ and $W(s) \leq w_{e_{h_t+1}} + w_{h_t+1}$. Let s denote the current solution. According to Lemma 6.1 and 6.2, if $W(s) \geq w_{e_{k+1}} + w_{k+1}$, removing any item from the second block improves the benefit function. Moreover, adding any item to the second block decreases the benefit. The probability of a one-bit flip, which removes an item from the second block, is $r/(2n)$ and it happens in expected time $O(2n/r)$. Since there are at most n items to be removed, RLS_swap obtains s with $W(s) < w_{e_{k+1}} + w_{k+1}$ in $2n\,(\sum_{i=1}^{r} 1/i) = O(n \log n)$ expected time. Note that after this point, no item can be removed from s with a one-bit flip.

Now we examine the value of y as defined in Equation 6.5. Let $y > h_t$. Since there is no accepted one-bit flip that removes an item, we have $W(s) < w_{e_y} + w_y$ and Corollary 6.4 holds. Otherwise, we have $y = h_t$ and there are two cases: $y = h_t = k$ or $y = h_t < k$. Let $y = h_t = k$. In this case, s includes of all the items in the first block and there is no item in the second block to be removed. Therefore, RLS_swap has found s^* which is the optimal solution. In the other case, that $y = h_t < k$, adding e_{h_t+1} is beneficial and there exists at least one acceptable one-bit flip that adds an item to the solution s in expected time $O(n)$. Let $y > h_t$ be the first item added. Since this bit flip is beneficial, we have $W(s) + w_y < w_{e_y} + w_y$ and Corollary 6.4 holds. Thus in $O(n)$ we have a solution in which $y > h_t$ and RLS_swap cannot remove e_y. At this stage, any two-bit flip that swaps e_y and e_{h_t+1} improves h_t by one. This swap takes place in $O(n^2)$ expected time. Thus in expected time $O(n^2)$ RLS_swap increases h_t by one. Since the maximum value of h_t is $k \leq n$, RLS_swap achieves $h_t = k$ in expected time $O(n^3)$.

Finally in the third phase, RLS_swap needs to remove the remaining items from the second block. Note that the first k items are now selected and cannot be removed anymore. Each item can be removed with a one-bit flip which results in $O(n \log n)$ expected time for this phase.

Therefore we can conclude that RLS_swap finds the optimal solution for the PWT with correlated weights and profits in $O(n^3)$ expected time. □

G-SEMO$_e$

In this section we consider the time performance of G-SEMO$_e$ on the PWT problem with correlated weights and profits. The two objectives used in this algorithm are the weight function and the lexicographical fitness function, denoted by $W(s)$ and $F(s)$, respectively. We say solution s_1 weakly dominates solution s_2, denoted by $s_1 \succeq s_2$, if $W(s_1) \leq W(s_2) \wedge F(s_1) \geq F(s_2)$. In the case of at least one strict inequality, it is called strong dominance. In our analysis, which is inspired by [NS18], we use a

fitness level argument on the weights of the solution and compute the expected time needed to find at least one Pareto solution. Next, we calculate the time for finding the entire Pareto front. Due to Lemma 6.2, we can observe the following corollary that describes the Pareto front structure.

Corollary 6.6. *The Pareto set corresponding to the Packing While Travelling problem with correlated weights and profits is the solution set* $\{s_0, \cdots, s_k = s^*\}$.

Proof. As it is explained in the proof of Lemma 6.1, s_i, $i \leq n$, dominates all the solutions with size i. On the other hand, we have $B(s_0) \leq \cdots \leq B(s_k)$ while $W(s_0) \leq \cdots \leq W(s_k)$, which implies that $\{s_0, \cdots, s_k\}$ do not dominate each other. Furthermore, s_k dominates every solution s_i, $i > k$, since it has a higher fitness value and less weight. Thus, there exists no solution dominating $\{s_0, \cdots, s_k = s^*\}$, which completes the proof. □

The following theorem proves that the expected time for G-SEMO$_e$ to find the entire Pareto front of the PWT problem with correlated weights and profits is $O(n^3)$.

Theorem 6.7. *G-SEMO$_e$ finds all the non-dominated solutions of the Packing While Travelling problem with correlated weights and profits in* $O(n^3)$ *expected time.*

Proof. The proof consists of two phases. In the first phase, G-SEMO$_e$ finds the Pareto solution $\{0\}^n$. In the second phase, G-SEMO$_e$ finds other Pareto solutions based on the assumption of having at least one optimal solution.

Note that according to the definition of $F(s)$, an infeasible solution with less constraint violation always dominates the other ones, even in this two-objective space. Hence, while G-SEMO$_e$ has not achieved a feasible solution, the size of its population remains one. Therefore, it behaves exactly the same as (1+1) EA until it finds a feasible solution, which is argued in first phase of Theorem 6.5, and takes $O(n^2)$ expected time. In the rest of the analysis of the first phase, we assume that G-SEMO$_e$ has found a feasible solution.

To analyse the first phase, we define n fitness levels A_i, $0 \leq i \leq n-1$, based on the weight objective $W(s)$ as follows:

$$\begin{aligned} A_i &= \{s \mid s = s_0\} && i = 0 \\ A_i &= \{s \mid W(s_{i-1}) < W(s) \leq W(s_i)\} && 1 \leq i \leq n. \end{aligned}$$

According to the correlation of the weights, this definition guarantees that if a solution s belongs to level A_i, $1 \leq i \leq n$, then s includes at least one of the items e_j where $j \geq i$. Moreover, removing e_j from s moves it to a lower level. Now assume that P denotes the population in step t of G-SEMO$_e$. Let A_u denote the lowest level that has been achieved until step t and $s \in A_u$ has the highest fitness among the solutions in level A_u. G-SEMO$_e$ chooses this solution by choosing $|s|_1$ from I, which happens

with the probability of $1/|I| \geq 1/n$. Furthermore, there exists item e_x in s such that $x \geq u$ and removing it produces a solution in a lower level. This solution will be accepted since it has the lowest weight. The one-bit flip that removes e_x from s happens with the probability of $1/(en)$. Thus, G-SEMO$_e$ reduces u with probability of $1/(en^2)$ in step t. In other words, G-SEMO$_e$ reduces u, at least by one, in expected time $O(n^2)$. On the other hand, s_0, which is the only solution of level A_0, is a Pareto solution since it has the lowest possible weight. Thus, if the algorithm achieves A_0 then the first phase is accomplished. Since the maximum possible value of u is n and u is reduced by one in every $O(n^2)$ expected iterations, G-SEMO$_e$ finishes the first phase in expected time of $O(n^3)$.

In the second phase, we assume $s_i \in P$, $i \leq k$ exists such that s_i is Pareto solution by Corollary 6.6, and either s_{i-1} or s_{i+1}, which are also Pareto solutions, do not exist in P. Otherwise, the second phase is already finished. Adding e_{i+1} to or removing e_i from s_i results in s_{i+1} or s_{i-1}, respectively. For such a step, the algorithm must choose $i \in I$ and flip the correct bit, which happens with the probability $1/n$ and $1/(en)$, respectively. Hence, the algorithm finds a new Pareto solution from s_i in $O(n^2)$ expected iterations. Based on Corollary 6.6, the size of the Pareto set is $k+1 \leq n$. Therefore, G-SEMO$_e$ finds all the $k+1$ Pareto solutions of the PWT problem with correlated weights and profits in $O(n^3)$ expected time. □

6.3.2 Uniform Weights

In this section, we analyse the performance of (1+1) EA on another version of the Packing While Travelling problem. Here we assume that weights of all the items are one and the profits are arbitrary. Similar to the previous instances, we assume items $\{e_1, \cdots, e_n\}$ are indexed such that $p_1 \geq \cdots \geq p_n$. Note that the results of Lemmata 6.1 and 6.2 also holds for the uniform weights. Moreover, since correlated weights and profits are actually the more general version of the uniform weights, the results for G-SEMO$_e$ and RLS_swap also holds.

The analysis of (1+1) EA is based on the following Lemma which proves the existence of a set of one-bit flips and two-bit flips which transform an arbitrary solution to the optimal solution.

Lemma 6.8. *Let $W(s^*)$ be the weight of the optimal solution and the current solution s is feasible. There exists a set of one-bit flips and two-bit flips that transform s to an optimal solution if they happen in any order.*

Proof. Assume that $W(s) \leq W(s^*)$. This implies that $i = |s|_1 \leq k$, where i is the number of selected items in s. In this case, any two-bit flips that swap a one bit from the second block and a zero bit from the first block will be accepted by the algorithm. If there is no one bit in the second block, then all the one-bit flips that change a zero bit in the first block are accepted and this set leads s to the optimal solution s^*. Note that all defined one-bit flips will be accepted (Lemma 6.1 and Lemma 6.2) since $i \leq k$

and the weights are one. Hence, it is not necessary for the bit flips to happen in a special order. The other case, that $C \geq W(s) > W(s^*)$, is similar to the first one. Any two-bit flips that swap a zero bits in the first block with a one bit in the second block and any one-bit flips that remove ones from the second block are accepted by the algorithm. Thus, in this case, there also exists a set of one-bit flips and two-bit flips that, if they occure in any order, transform s into the optimal solution. □

Here we present some more definitions that help us with analysis of the performance of (1+1) EA on the PWT problem. Assume M is the set of m_1 one-bit flips and m_2 two-bit flips that transform the current solution s to an optimal solution and $|M| = m_1 + m_2$. Moreover, let $g(s) = B(s^*) - B(s)$ be the difference between the benefit of s and the optimal solution. Therefore, we can denote the contribution of one-bit flips and two-bit flips in $g(s)$ by $g_1(s)$ and $g_2(s)$, respectively, such that $g(s) = g_1(s) + g_2(s)$. In the next theorem, we calculate the expected time for (1+1) EA to find the optimal solution of the PWT problem with unifrom weights.

Theorem 6.9. *The expected time for (1+1) EA to find the optimal solution for the Packing While Travelling problem with uniform weights is* $O(n^2 \max\{\log n, \log p_{\max}\})$.

Proof. As mentioned in the proof of Theorem 6.5, it is proven in Theorem 3 in [NS18] that (1+1) EA finds a feasible solution in $O(n^2)$ expected time. Thus, we assume that (1+1) EA has already found a feasible solution s. Let $\Delta_t = g(s) - g(s') = B(s') - B(s)$ denote the improvement of (1+1) EA in iteration t which transforms s to s'. We partition the proof into two cases. Firstly, let the overall contribution of one-bit flips in $g(s)$ be more than the total improvement that could be achieved by two-bit flips. Hence, we have $g_1(s) \geq \frac{g(s)}{2}$. Since there are m_1 one-bit flips in M, each of them happens with the probability of $\frac{m_1}{en}$ and improves the solution in average by $\frac{g_1(s)}{m_1}$. Thus, in this case we have :

$$E[\Delta_t] \geq \frac{g_1(s)}{m_1} \cdot \frac{m_1}{en} \geq \frac{g_1(s)}{en} \geq \frac{g(s)}{2en}.$$

In the second case, the sum of improvements by two-bit flips in M is more than the total improvement of one-bit flips and we have $g_2(s) \geq \frac{g(s)}{2}$. Here, the average improvement of each two-bit flip is $\frac{g_2(s)}{m_2}$ and each takes place with a probability of $\frac{m_2}{2en^2}$. Therefore, we have:

$$E[\Delta_t] \geq \frac{g_2(s)}{m_2} \cdot \frac{m_2}{2en^2} \geq \frac{g_2(s)}{2en^2} \geq \frac{g(s)}{4en^2}.$$

We can conclude that the expected improvement at step t is at least $E[\Delta_t] \geq \frac{g(s)}{4en^2}$. On the other hand, $B(s^*) \leq n \cdot p_{\max}$ is the maximum possible benefit value, ignoring the cost function. To use the multiplicative drift, it is only necessary to calculate the minimum possible amount of $g(s)$.

Let s^l be the last solution that turned into an optimal solution with a bit flip. We need to find the minimum of $B(s^*) - B(s^l)$ which happens when $B(s^l)$ is maximized. There are three possibilities in which s^l is the closest solution to the optimal solution. If the last bit flip is a two-bit flip, to maximize $B(s^l)$, $s^l = s_{k-1} \cup e_{k+1}$. In this case, we have $g(s^l) = p_k - p_{k+1} \geq 1$, since all the profits are integers. If the last bit flip is a one-bit and it adds an item to s^l, then the maximum benefit of s^l is achieved when $s^l = s_{k-1}$. Therefore, we have $g(s^l) = p_k - \frac{Rdv}{(v_{\max} - v(k-1))(v_{\max} - vk)}$. Considering $R, d, v_{\min}$ and $v_{\max}$ as constants, $g(s) = O(n^{-q})$ for some constant $q \geq 1$. The same result holds for the third case where the final bit flip removes e_{k+1} from $s^l = s_{k+1}$.

Finally, using the multiplicative drift with $X_0 = n \cdot p_{\max}$, $x_{\min} = n^{-q}$ and $\delta = \frac{1}{4en^2}$, the expected first hitting time T that (1+1) EA finds the optimal solution for the PWT problem with uniform weights is:

$$E[T|X_0] \leq 4en^2 \ln(n \cdot p_{\max} \cdot n^q) = O(n^2 \max\{\log n, \log p_{\max}\}).$$

□

6.4 Experiments

In this section, we experimentally investigate the performance of our algorithms. The goal is to complement the theoretical analysis to gain additional insight into the behaviour of the algorithms during the optimisation process. Moreover, by analysing the running time of our algorithms on random instances with different sizes, we present a clearer view of how they perform.

In the previous section, we showed that the classical RLS cannot find the optimal solution by doing only one-bit flips and it is essential to do a two-bit flip to escape local optima. However, what happens if the classical RLS is enhaced with a population? To answer this question, we also consider another two multi-objective algorithms called SEMO_e and SEMO_swap. They are different from G-SEMO_e (Algorithm 9) only in the mutation step. The mutation step in SEMO_swap is the same as Algorithm 13. SEMO_e, on the other hand, only chooses one bit, uniformly at random, and flips it. Our experiments show that SEMO_e avoids the classical RLS weakness by using the population and performs even better than the other multi-objective algorithms.

6.4.1 Benchmarking and Experimental Setting

To compare the practical performance of different algorithms in different types of PWT, we used thirty different instances where each instance consists of 300 items. Each instance of size n for the correlated PWT is generated by choosing n integer profits uniformly at random within [1,1000] and assigning them to $p_1, \cdots, p_n$ in descending order. Similarly, n uniformly random integers within the same interval are

generated and sorted in ascending order as the weights $w_1, \cdots, w_n$. Hence, for any item $e_i, e_j : i < j$ we have $p_i \geq p_i$ and $w_i \leq w_j$. For the instances with uniform weights, we use the same profits as correlated instances but change the weights to one. We set the constant values in all the instances as follows: $n = 300$, $d = 50$, $R = 70$, $v_{\max} = 1$ and $v_{\min} = 0.1$. $C = 8000$ is the capacity chosen for correlated instances. To calculate the proper capacity for uniform instances, we use the average of the maximum number of items with correlated weights that fit in $C = 8000$, which results in $C = 72$ for the uniform instances. Furthermore, for all the experiments, algorithms start with the zero solution.

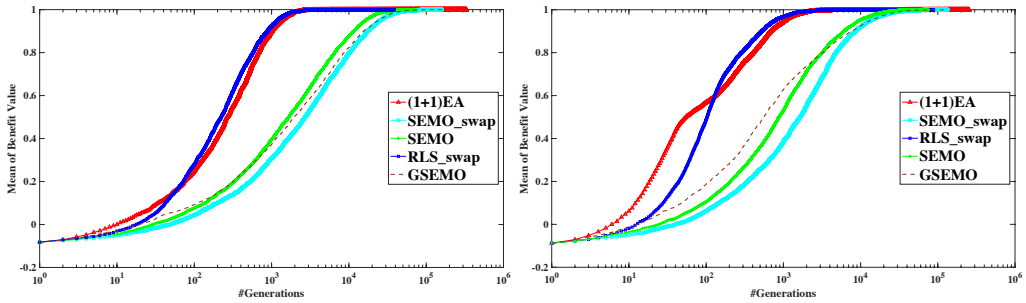
We use these instances in the first experiment, in which we show how the algorithms converge to the optimal solution for the correlated and uniform instances. We run each algorithm for all thirty instances, record the best found solution in each generation and normalise its benefit value to the interval [0,1] with respect to the optimal solution for each instance. We plot the average of normalised values as the success rate for each algorithm (Figure 6.1). Hence, this value is one when an algorithm finds the optimal solution for all thirty different instances.

In the final experiments, in which we consider the optimisation time for algorithms, we run each algorithm on instances with seven different sizes, $n = 100, 200,$ $500, 1000, 2000, 5000, 10000$. For each n, we have thirty different instances which are created with the same constants as the previous ones. For each algorithm, we record the average number of evaluations to find the optimal solution for the thirty instances with the same n as its optimisation time. Hence, for each algorithm, we have seven points to estimate the performance (Figure 6.2). We only do this analysis with the correlated instances.

In our results for (1+1) EA and G-SEMO$_e$, we only count the generations that an actual bit flip has happened in mutation steps. The same approach has been used in [DW18]. This causes the analysis to be fair since RLS_swap guarantees to create a new solution in each mutation step but (1+1) EA and G-SEMO$_e$ do not flip any of the bits with the probability of $1/e$ (36.7% of times).

6.4.2 Analysis

In this section, we analyse the performance of the algorithms based on the experimental results. Figure 6.1 illustrates how the algorithms converge to the optimal solutions for all the instances on average. In both types of the instances, it can be observed that (1+1) EA and RLS get close to the optimal solutions much faster than the multi-objective algorithms, which demonstrates that using population slows down the convergence rate significantly. (1+1) EA, however, has major problems in finding the optimal solutions when it is close to the optimum and only a few bit flips are needed. The reason is that (1+1) EA is able to flip more than one bit. Hence, it is likely to improve the benefit value, but increase the hamming distance between the



(A) Correlated weights and profits (B) Uniform weights

FIGURE 6.1: The average best solution in each generation for RLS_swap, (1+1) EA, G-SEMO$_e$, SEMO$_e$ and SEMO_swap in thirty different instances.

current solution and the optimal solution in the same time. In other words, (1+1) EA is able to improve the benefit function while the improved solution needs more bit flips to reach the optimal solution than before. Note that Figure 6.1 is plotted with a logarithmic x scale to make it easier to distinguish the difference between the first 10^5 generations. While the other algorithms almost achieve the optimal solution in the first 10^5 generations, (1+1) EA performs two times worse than the others.

Looking into the multi-objective algorithms, Figure 6.1 illustrates that G-SEMO$_e$ and SEMO_swap behave almost identically while SEMO$_e$ outperforms both. This shows that although SEMO$_e$ can flip only one bit in each generation, the population enables the algorithm to bypass the local optima and find a better solution. In other words, using population and dominance concepts is another method to escape local optima with only one-bit flips. The reason, as described in the proof of Theorem 3.7, is that one-bit flips can always reduce the weight of the solution. Using weight as an objective in population-based multi-objective algorithms makes it possible for SEMO$_e$ to obtain the zero solution, which is a Pareto optimal solution, by using only one-bit flips. From that point, the algorithm can gradually find all the other Pareto

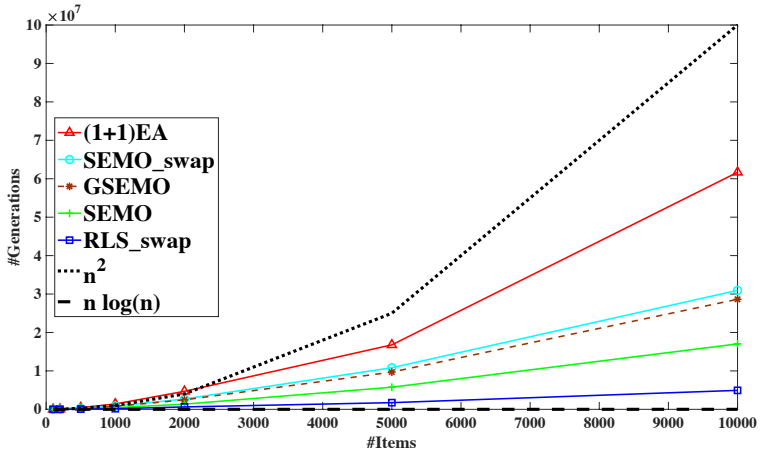


FIGURE 6.2: Comparison between the running time of the algorithms according to the experimental results.

optimal solutions, including the general optimal solution, by adding the correct item to the previous optimal solution. Generally speaking, RLS_swap and SEMO_e, which enhance versions of the classical RLS to avoid the local optima, perform better for this Packing While Travelling problem variation.

The results of the final experiment are presented in Figure 6.2. We run each algorithm for instances with seven different sizes, and for each specific size, we have thirty different instances. Figure 6.2 shows the average of running times for each instance size. It validates the results in Figure 6.1, in which RLS_swap has a better performance than the other algorithms while multi-objective algorithms outperform (1+1) EA. This data can also be used to give an insight into the expected optimisation time for each algorithm. Figure 6.2 presents the data for n^2 and $n \log n$ expressions which suggests that the order of magnitude for the running times for these algorithms on random instances is close to n^2 or $n \log n$.

6.5 Conclusion

While evolutionary algorithms have been thoroughly studied in solving linear functions, their performance on non-linear problems is not clear. In this chapter, we study the performance of three base-line EAs, from a theoretical viewpoint, on the packing while travelling problem, which is also known as a non-linear knapsack problem. We prove that RLS_swap and G-SEMO_e find the optimal solution in $O(n^3)$ expected time for instances with correlated weights and profits. In addition, we show that (1+1) EA finds the optimal solution for instances with uniform weights in $O(n^2 \log(\max\{n, p_{\max}\}))$, where $p_{\max}$ is the highest profit of the given items. We also empirically investigate these algorithms and, based on our investigations, we conjecture an upper bound of $O(n^2)$.

Part III

Dynamic Combinatorial Optimisation Problems

Chapter 7

Evolutionary Multi-Objective Optimisation for the Dynamic Knapsack Problem

7.1 Introduction

Evolutionary algorithms and other bio-inspired computing have been applied to many dynamic and stochastic problems [NY12; RKD17] as they have the ability to easily adapt to changing environments.

Theoretical investigations for combinatorial optimisation problems with dynamically changing constraints have been carried out on different problems [NPR20], recently. Studies investigated the efficiency of algorithms in finding good quality solutions from scratch, when criteria change dynamically during optimisation process, and/or adopting the current valuable solution when a new dynamic change happens. The goal of this chapter is to contribute to this research direction from an experimental perspective.

The multidimensional knapsack problem has been previously investigated in a dynamic environment [UU09; BÖ14; BÖ17]. In this problem, each item has m profits and weights in m dimensions. The capacity of knapsack has also been defined in an m dimensional space, and the goal is to maximise total profit without exceeding the capacity in any of the dimensions. Baykosoglu and Ozsoydan in [BÖ17] compared the performance of evolutionary and population-based methods, such as differential evolution (DE), memetic algorithm (MA), and firefly algorithm (FA), versus constructive search strategies such as greedy randomised adaptive search procedure (GRASP). They considered two different scenarios. Firstly, they fixed the magnitude of changes and compared algorithms in various alteration frequencies. They showed that FA and GRASP perform significantly better in this scenario. However, in the scenario with various magnitudes, constructive strategies seem to be more sensitive.

In this chapter, we use dynamic version of one dimensional knapsack problem and compare the algorithms in more challenging scenarios in which the magnitude, the distribution, and frequency of changes vary.

To experimentally investigate evolutionary algorithms for the knapsack problem with dynamically changing capacity, we design a specific benchmark set. This benchmark set is built on classical static knapsack instances and varies the constraint bound over time. The changes of the constraint bound occur randomly every τ iterations, where τ is a parameter determining the frequency of changes. The magnitude of a change is either chosen according to a uniform distribution in an interval $[-r, r]$, where r determines the magnitude of changes. Furthermore, we examine changes according to the normal distribution $\mathcal{N}(0, \sigma^2)$ with mean 0 and standard deviation σ. Here σ is used to determine the magnitude of changes and large values of σ make larger changes more likely.

The comparison between the algorithms is based on the offline errors. We compute the exact optimal solutions for each possible capacity by performing dynamic programming in preprocessing phase. To calculate the total offline error, we consider the difference between the profit of the best achieved feasible solution in each iteration and the profit of the optimal solution. Total offline error illustrates the performance of the algorithms during the optimisation process and demonstrate how fast an algorithm finds a good quality feasible solution. In the second part of the chapter, which considers advanced evolutionary algorithms, we also do the comparisons according to the partial offline error. The partial offline error only considers the best feasible solution found by an algorithm exactly before the next dynamic change happens. This factor does not illustrate how fast the algorithm finds a good quality solution. In this way, instead of analysing the performance during the optimisation period, we study the algorithms based on their final results.

The first part of our experimental analysis, which investigates the theoretical results in [Shi+19], corresponds to examining the uniform instances, in which all the weights are one. We consider the performance of (1+1) EA (Algorithm 3) against two versions of multi-objective evolutionary algorithms (MOEA and MOEA_D). The multi-objective approaches, which only differ in their definition of dominance, are developed based on the G-SEMO (Algorithm 4) in a way that store infeasible solutions as part of the population in addition to feasible solutions. The range of feasible and infeasible solutions stored in the multi-objective algorithms is set based on the anticipated change of the constraint bound. Our experimental analysis confirm the theoretical results and show that the multi-objective approaches that use population to deal with the dynamic changes outperform (1+1) EA.

Afterwards, we study the performance of the same baseline algorithms dealing with

general version of the dynamic knapsack problem, in which the weights are not uniform. For the general setting, we investigate different instances of knapsack problem, such as instances with randomly chosen weights and profits, and instances with strongly correlated weights and profits. We study the behaviour of the algorithms in various situations, where the frequency and the magnitude of changes are different. Our results show that the (1+1) EA has an advantage over the multi-objective algorithms when the frequency of changes is high. In this case, the population slow down the adaptation to the changes. On the other hand, lower frequency of changes play in favor of the multi-objective approaches, in most of the cases. Exceptions occur in some situations where weights and profits are highly correlated or have similar values.

In addition to these baseline evolutionary algorithms, we extend our experiments to investigating the performance of NSGA-II and SPEA2 (Algorithms 5 and 6 of Section 7), as the representatives of advanced evolutionary algorithms, in the dynamic environment. We use Jmetal package as the base of our implementations and modify the algorithms to perform on the dynamic KP [DN11]. Each of the algorithms calculate a specific fitness value based on the non-dominance rank and position of each individual among the others in the objective space. We compare these algorithms with MOEA_D, as it outperforms the other baseline evolutionary algorithms, and investigate the performance of advanced techniques in NSGA-II and SPEA2 against the simple approach of MOEA_D. Our experimental results illustrates that while NSGA-II and SPEA2 react faster to a dynamic change, MOEA_D can find better solution if it has enough time before the next dynamic change. We also show that the techniques for producing well-distributed non-dominated solutions prevent these algorithms to improve their best feasible solution. Thus, we address this problem by presenting an additional elitism to address this problem.

The work of this chapter is based on a conference paper [RNN18] presented at the 15th international conference on parallel problem solving from nature (PPSN XV) and its extended version that is submitted to the evolutionary computation journal [RNN20]. The outline of the chapter is as follows: Section 7.2 introduces the knapsack problem, how the dynamism is applied and our benchmarks. The baseline evolutionary algorithms and the detailed analysis of their experimental results is presented in Section 7.3. In Section 7.4, we present NSGA-II and SPEA2, the necessary modifications to apply them on the dynamic knapsack problem, and their experimental results in detail. Finally, a conclusion follows in Section 7.5.

7.2 The Dynamic Knapsack Problem

In this section, we explain how a dynamic change impacts the constraint bound in KP (knapsack problem introduced in Section 3.3), and introduce the details of benchmarks and the experimental settings used in Sections 7.3 and 7.4.

We consider two types of this problem based on the consideration of the weights. Firstly, we assume that all the weights are one and uniform dynamic constraint is applied. In this case, the limitation is on the number of items chosen for each solution and the optimal solution is to pick C items with the highest profits. Next, we consider the general case where the profits and weights are linear integers under linear constraint on the weight.

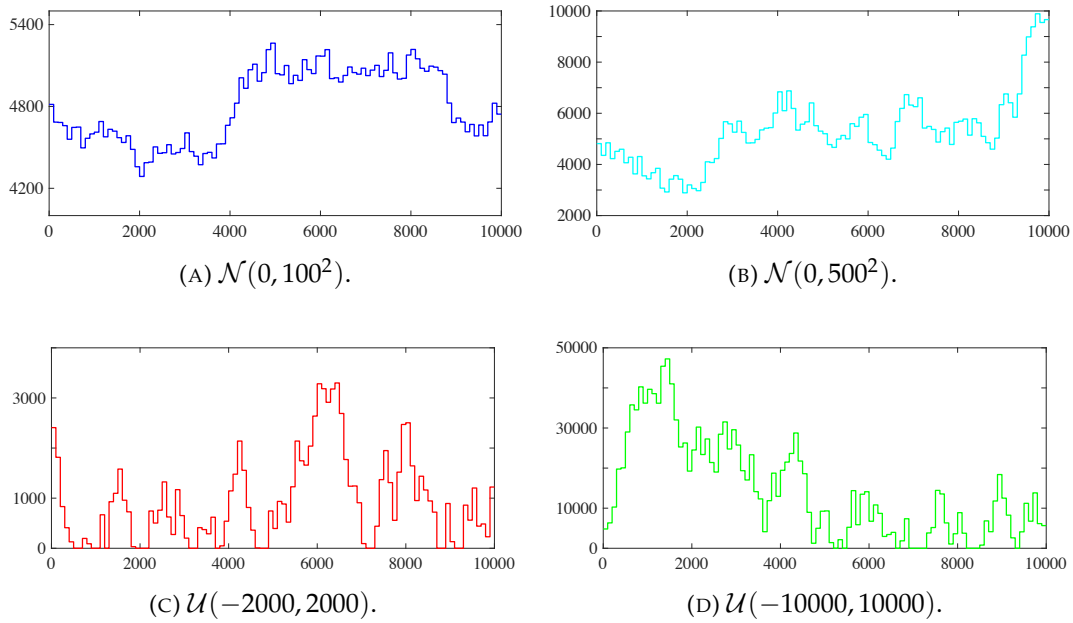
7.2.1 The Dynamic Constraint

In the dynamic version of KP considered in this chapter, the capacity dynamically changes during the optimisation with a preset frequency factor denoted by τ. A change happens every τ generations, i.e., the algorithm has τ generations to find the optimum of the current capacity and to prepare for the next change. In the case of uniformly random alterations, the capacity of next interval is achieved by adding a uniformly random value in $[-r, r]$ to C. Moreover, we consider another case in which the amount of the changes is chosen from the Gaussian distribution $\mathcal{N}(0, \sigma^2)$. Figure 7.1 illustrates how dynamic changes from different distributions affect the capacity. Note that the scales of the subfigures are not the same. For example, the total change after 100 dynamic changes under $\mathcal{N}(0, 100^2)$ is less than 1000 (Figure 7.1a) while the capacity reached almost 45000 with dynamic changes under $\mathcal{U}(-10000, 10000)$ (Figure 7.1d). This indicates that there are different types of challenges, resulting from the dynamic changes that the algorithms must consider.

The combination of different distributions and frequencies brings interesting challenges for the algorithms. In an environment where the constraint changes with a high frequency, the algorithms have less time to find the optimal solution, hence, it is likely that an algorithm which tries to improve only one solution will perform better than another algorithm that needs to optimise among several solutions. On the other hand, the algorithms that pay a lot of attention to the configuration of solutions in objective space, might loose the track of new optimal solution. This is caused by their preference in solutions with better distribution factor, instead searching for feasible solutions that are closer to the capacity constraint. Furthermore, the uniform distribution guarantees upper and lower bounds on the magnitude of the changes. This property could be beneficial for the algorithms which keep a certain number of solutions in each generation, so that they do get ready and react faster after a dynamic change. If the changes happen under a normal distribution, however, there is no strict bound on the value of any particular change, which means it is not easy to predict which algorithms will perform better in this type of environment.

7.2.2 Benchmark and Experimental Setting

In this section, we present the dynamic benchmarks and the experimental settings. We use eil101 benchmarks, which were originally generated for Traveling Thief Problem [Pol+14], ignoring the cities and only using the items. The weights and



(A) $\mathcal{N}(0, 100^2)$. (B) $\mathcal{N}(0, 500^2)$.

(C) $\mathcal{U}(-2000, 2000)$. (D) $\mathcal{U}(-10000, 10000)$.

FIGURE 7.1: Examples for constraint bound C over 10000 generations with $\tau = 100$ using uniform and normal distributions. Initial value $C = 4815$.

profits are generated in three different classes. In Uncorrelated (uncorr) instances, the weights and profits are integers chosen uniformly at random within $[1, 1000]$. The Uncorrelated Similar Weights (unc-s-w) instances have uniformly distributed random integers as the weights and profits within $[1000, 1010]$ and $[1, 1000]$, respectively. Finally, there is the Bounded Strongly Correlated (bou-s-c) variations which result in the hardest instances and comes from the bounded knapsack problem. The weights of this instance are chosen uniformly at random within $[1, 1000]$ and the profits are set according to the weights within the weights plus 100. In addition, in Section 7.3.2, where the weights are one, we set all the weights to one and consider the profits as they are in the benchmarks. The initial capacity in this version is calculated by dividing the original capacity by the average of the profits. Dynamic changes add a value to C each τ generations. Four different situations in terms of frequencies are considered: high frequent changes with $\tau = 100$, medium frequent changes with $\tau = 1000$, $\tau = 5000$ and low frequent changes with $\tau = 15000$.

In the case that weights are 1, the value of dynamic changes are chosen uniformly at random within the interval $[-r, r]$, where $r = 1$ are $r = 10$. In the case of linear weights, when changes are uniformly random, we investigate two values for r: $r = 2000, 10000$. Also, changes from normal distribution is experimented for $\sigma = 100$, $\sigma = 500$.

We use the offline errors to compute the performance of the algorithms. In each generation, we record error $e_i = p(x_i^*) - p(x_i)$ where x_i^* and x_i are the optimal solution and the best achieved feasible solution in generation i, respectively. If no feasible solution is found in generation i, we consider solution y_i that denotes the

solution with the smallest constraint violation. Then, the offline error is calculated as $e_i = p(x_i^*) + \nu(y_i)$. Hence, the total offline error for m generations would be $\sum_{i=1}^{m} e_i/m$.

The benchmarks for dynamic changes are thirty different files. Each file consists of 100000 changes, as numbers in $[-r, r]$ generated uniformly at random. Similarly, there are thirty other files with 100000 numbers generated under the normal distribution $\mathcal{N}(0, \sigma^2)$. The algorithms start from the beginning of each file and pick the number of change values from the files. Hence, for each setting, we run the algorithms thirty times with different dynamic change values and record the total offline error of each run.

In order to establish a statistical comparison of the results among different algorithms, we use a multiple comparisons test. In particularity, we focus on the method that compares a set of algorithms. For statistical validation we use the Kruskal-Wallis test with 95% confidence. Afterwards, we apply the Bonferroni post-hoc statistical procedures that are used for multiple comparisons of a control algorithm against two or more other algorithms. For more detailed descriptions of the statistical tests we refer the reader to [CF09].

Our results are summarised in the Tables 7.1 .. 7.7. The columns represent the algorithms with the corresponding mean value and standard deviation. Note, $X^{(+)}$ is equivalent to the statement that the algorithm in the column outperformed algorithm X, and $X^{(-)}$ is equivalent to the statement that X outperformed the algorithm in the given column. If the algorithm X does not appear, this means that no significant difference was observed between the algorithms.

7.3 Baseline Evolutionary Algorithms

In this section, we introduce the baseline evolutionary algorithms that are considered in this chapter and present detailed comparison of their performance according the total offline values.

7.3.1 Algorithms

We investigate the performance of three algorithms in this section. The initial solution for all these algorithms is a solution with items chosen uniformly at random. After a dynamic change happens to constraint C, all the algorithms update the solution(s) and start the optimisation process with the new capacity. This update is addressing the issue that after a dynamic change, current solutions may become infeasible or the distance of its weight from the new capacity become such that it is not worth to be kept anymore. (1+1) EA (Algorithm 3) flips each bit of the current solution with the probability of $\frac{1}{n}$ as the mutation step. Afterward, the algorithm chooses between the original solution and the mutated one using the value of the

Algorithm 15: MOEA

Update C;
$S^+ \leftarrow \{z \in S^+ \cup S^- | C < w(z) \leq C + \delta\}$;
$S^- \leftarrow \{z \in S^+ \cup S^- | C - \delta \leq w(z) \leq C\}$;
if $S^+ \cup S^- = \emptyset$ **then**
    $q \leftarrow$ best previous solution;
if $C < w(q) \leq C + \delta$ **then**
    $S^+ \leftarrow \{q\} \cup S^+$;
else if $C - \delta \leq w(q) \leq C$ **then**
    $S^- \leftarrow \{q\} \cup S^-$;
while *a change happens* **do**
    if $S^+ \cup S^- = \emptyset$ **then**
        Initialise S^+ and S^- by Repair(q,δ,C);
    else
        choose $x \in S^+ \cup S^-$ uniformly at random;
        $y \leftarrow$ flip each bit of x independently with probability $\frac{1}{n}$;
        if $(C < w(y) \leq C + \delta) \wedge (\nexists p \in S^+ : p \succcurlyeq_{MOEA} y)$ **then**
            $S^+ \leftarrow (S^+ \cup \{y\}) \setminus \{z \in S^+ | y \succ_{MOEA} z\}$;
        if $(C - \delta \leq w(y) \leq C) \wedge (\nexists p \in S^- : p \succcurlyeq_{MOEA} y)$ **then**
            $S^- \leftarrow (S^- \cup \{y\}) \setminus \{z \in S^- | y \succ_{MOEA} z\}$;

fitness function. Let $p_{max} = \max_{1 \leq i \leq n} p_i$ be the maximum profit among all the items. The fitness function that we use in (1+1) EA is as follows:

$$f_{1+1}(x) = p(x) - (n \cdot p_{max} + 1) \cdot \nu(x),$$

where $\nu(x) = \max\{0, w(x) - C\}$ is the constraint violation of x. If x is a feasible solution, then $w(x) \leq C$ and $\nu(x) = 0$. Otherwise, $\nu(x)$ is the weight distance of $w(x)$ from C.

The algorithm aims to maximise f_{1+1} which consists of two terms. The first term is the total profit of the chosen items and the second term is the applied penalty to infeasible solutions. The amount of penalty guarantees that a feasible solution always dominates an infeasible solution. Moreover, between two infeasible solutions, the one with weight closer to C dominates the other one.

The other algorithm we consider in this chapter is a multi-objective evolutionary algorithm (Algorithm 15), which is inspired by a theoretical study on the performance of evolutionary algorithms in the reoptimisation of linear functions under dynamic uniform constraints [Shi+19]. Each solution x in the objective space is a two-dimensional point $f_{MOEA}(x) = (w(x), p(x))$. We say solution y dominates solution x w.r.t. f_{MOEA}, denoted by $y \succcurlyeq_{MOEA} x$, if $w(y) = w(x) \wedge f_{(1+1)}(y) \geq f_{(1+1)}(x)$.

According to the definition of $\succcurlyeq_{MOEA}$, two solutions are comparable only if they have the same weight. Note that if x and y are infeasible and comparable, then the

Algorithm 16: Repair

input : Initial solution q, δ, C
output: S^+ and S^- such that $|S^+ \cup S^-| = 1$

```
1 while |S+ ∪ S−| = 0 do
2     y ← flip each bit of q independently with probability of 1/n;
3     if f_{1+1}(y) ≥ f_{1+1}(q) then
4         q ← y;
5         if C < w(q) ≤ C + δ then
6             S+ ← {q} ∪ S+;
7         else if C − δ ≤ w(q) ≤ C then
8             S− ← {q} ∪ S−;
```

one with higher profit dominates. MOEA uses a parameter denoted by δ, which determines the maximum number of individuals that the algorithm is allowed to store around the current C. For any weight in $[C - \delta, C + \delta]$, MOEA keeps a solution. The algorithm prepares for the dynamic changes by storing nearby solutions, even if they are infeasible as they may become feasible after the next change. A large δ, however, causes a large number of solutions to be kept, which reduces the probability of choosing anyone. Since the algorithm chooses only one solution to mutate in each iteration, this affects the MOEA's performance in finding the optimal solution.

After each dynamic change, MOEA updates the sets of solutions. If a change occurs such that all the current stored solutions are outside of the storing range, namely $[C - \delta, C + \delta]$, then the algorithm consider the previous best solution as the initial solution and uses the Repair function (Algorithm 16), which behaves similar to (1+1) EA, until a solution with weight distance δ from C is found.

To address the slow rate of improvement of MOEA caused by a large δ, we use the standard definition of dominance in multi-objective optimisation– i.e., solution y dominates solution x, denoted by $y \succcurlyeq_{MOEA\_D} x$, if $w(y) \leq w(x) \wedge p(y) \geq p(x)$. This new algorithm, called MOEA_D, is obtained by replacing lines 14-19 of Algorithm 15 with Algorithm 17. It should be noticed that if y is an infeasible solution then it is only compared with other infeasible solutions and if y is feasible it is only compared with other feasible solutions. MOEA_D keeps fewer solutions than MOEA and overall the quality of the kept solutions is higher, since they are not-dominated by any other solution in the population.

7.3.2 Experimental Results

In this section we describe the initial settings of the algorithms and analyse their performance using the mentioned statistical tests. The initial solution for all the algorithms is a pack of items which are chosen uniformly at random. Each algorithm initially runs for 10000 generations without any dynamic change. After this, the first change is introduced, and the algorithms run one million further generations with

Algorithm 17: MOEA_D (Dominance and Selection)

choose $x \in S^+ \cup S^-$ uniformly at random;
$y \leftarrow$ flip each bit of x independently with probability $\frac{1}{n}$;
if $(C < w(y) \leq C + \delta) \wedge (\nexists p \in S^+ : p \succcurlyeq_{MOEA\_D} y)$ **then**
  $S^+ \leftarrow (S^+ \cup \{y\}) \setminus \{z \in S^+ | y \succ_{MOEA\_D} z\}$;
if $(C - \delta \leq w(y) \leq C) \wedge (\nexists p \in S^- : p \succcurlyeq_{MOEA\_D} y)$ **then**
  $S^- \leftarrow (S^- \cup \{y\}) \setminus \{z \in S^- | y \succ_{MOEA\_D} z\}$;

TABLE 7.1: The mean, standard deviation values and statistical tests of the offline error for (1+1) EA, MOEA, MOEA_D based on the uniform distribution with all the weights as one.

	n	r	τ	(1+1) EA (1)			MOEA (2)			MOEA_D (3)		
				mean	st	stat	mean	st	stat	mean	st	stat
uncor	100	5	100	4889.39	144.42	$2^{(-)},3^{(-)}$	1530.00	120.76	$1^{(+)}$	1486.85	123.00	$1^{(+)}$
	100	5	1000	1194.23	86.52	$2^{(-)},3^{(-)}$	44.75	8.96	$1^{(+)}$	46.69	8.51	$1^{(+)}$
unc-s-w	100	5	100	4990.80	144.87	$2^{(-)},3^{(-)}$	1545.36	115.15	$1^{(+)}$	1500.07	106.70	$1^{(+)}$
	100	5	1000	1160.23	130.32	$2^{(-)},3^{(-)}$	41.90	6.13	$1^{(+)}$	43.06	7.22	$1^{(+)}$
bou-s-c	100	5	100	13021.98	780.76	$2^{(-)},3^{(-)}$	4258.53	580.77	$1^{(+)}$	4190.55	573.13	$1^{(+)}$
	100	5	1000	3874.76	911.50	$2^{(-)},3^{(-)}$	177.62	83.16	$1^{(+)}$	175.14	80.73	$1^{(+)}$

dynamic changes in every τ generations. For MOEA and MOEA_D, it is necessary to initially provide a value for δ. These algorithms keep at most δ feasible solutions and δ infeasible solutions, to help them efficiently deal with a dynamic change. When the dynamic changes come from $\mathcal{U}(-r, r)$, it is known that the capacity will change at most r. Hence, we set $\delta = r$. In case of changes from $\mathcal{N}(0, \sigma^2)$, δ is set to 2σ, since 95% of values will be within 2σ of the mean value. Note that a larger δ value increases the population size of the algorithms and there is a trade-off between the size of the population and the speed of algorithm in reacting to the next change.

Dynamic Uniform Constraint

In this section, we validate the theoretical results against the performance of (1+1) EA and Multi-Objective Evolutionary Algorithm. Shi et al. state that the multi-objective approach performs better than (1+1) EA in re-optimising the optimal solution of dynamic KP under uniform constraint [Shi+19]. Although the MOEA that we used in this experiment is not identical to the multi-objective algorithm studied previously in [Shi+19] and they only considered the re-optimisation time, the experiments show that multi-objective approaches outperform (1+1) EA in the case of uniform constraints (Table 7.1). An important reason for this remarkable performance is the relation between optimal solutions in different weights. In this type of constraint, the difference between the optimal solution of weight w and $w + 1$ is one item. As a result of this, keeping non-dominated solutions near the constrained bound helps the algorithm to find the current optimum more efficiently and react faster after a dynamic change.

Furthermore, according to the results, there is no significant difference between using MOEA and MOEA_D in this type of KP. Considering the experiments in Section 7.3.2, a possible reason is that the size of population in MOEA remains small when weights are one. Hence, MOEA_D, which stores fewer items because of its dominance definition, has no advantage in this manner anymore. In addition, the constraint is actually on the number of the items. Thus, both definitions for dominance result the same in many cases.

Dynamic Linear Constraint

In this section, we consider the same algorithms in more difficult environments where weights are arbitrary under dynamic linear constraint. As it is shown in Section 7.3.2, the multi-objective approaches outperform (1+1) EA in the case that weights are one. Now we try to answer the question: Does the relationship between the algorithms hold when the weights are arbitrary?

The data in Table 7.2 shows the experimental results in the case of dynamic linear constraints and changes under a uniform distribution. It can be observed that (as expected) the mean of errors decreases as τ increases. Larger τ values give more time to the algorithm to get closer to the optimal solution. Moreover, starting from a solution which is near to the optimal for the previous capacity, can help to speed up the process of finding the new optimal solution in many cases.

We first consider the results of dynamic changes under the uniform distribution. We observe in Table 7.2 that unlike with uniform constraint, in almost all the settings, MOEA has the worst performance of all the algorithms. The first reason for this might be that, in case of the uniform constraints, the magnitude of a capacity change is equal to the Hamming distance of optimal solutions before and after the change. In other words, when weights are one, we can achieve the optimal solution for weight w by adding an item to the optimal solution for weight $w-1$ or by deleting an item from the optimal solution for $w+1$. However, in case of arbitrary weights, the optimal solutions of weight w and $w+d$ could have completely different items, even if d is small. Another reason could be the effect of having a large population. A large population may cause the optimisation process to take longer and it could get worse because of the definition of $\succcurlyeq_{MOEA}$, which only compares solutions with equal weights. If s is a new solution and there is no solution with $w(s)$ in the set of existing solutions, MOEA keeps s whether s is a good solution or not, i.e., regardless of whether it is really a non-dominated solution or whether there exist another solution with lower weight and higher profit in the set. This comparison also does not consider if s has any good properties to be inherited by the next generation. For example, MOEA generate s that includes items with higher weights and lower profits. Since it might have a unique weight, MOEA keeps it in the population. Putting s in the set of solutions decreases the probability of choosing all other solution, even those solutions that are very close to the optimal solution. As

TABLE 7.2: The mean, standard deviation values and statistical tests of the offline error for (1+1) EA, MOEA, MOEA_D based on the uniform distribution.

	n	r	τ	(1+1) EA (1)			MOEA (2)			MOEA_D (3)		
				mean	st	stat	mean	st	stat	mean	st	stat
uncor	100	2000	100	5564.37	463.39	$2^{(+)},3^{(-)}$	11386.40	769.77	$1^{(-)},3^{(-)}$	3684.26	525.50	$1^{(+)},2^{(+)}$
	100	2000	1000	2365.56	403.64	$2^{(+)},3^{(-)}$	7219.17	587.50	$1^{(-)},3^{(-)}$	776.14	334.69	$1^{(+)},2^{(+)}$
	100	2000	5000	1415.42	167.08	$2^{(+)},3^{(-)}$	3598.29	420.12	$1^{(-)},3^{(-)}$	270.90	121.43	$1^{(+)},2^{(+)}$
	100	2000	15000	914.55	102.82	$2^{(+)},3^{(-)}$	2004.16	368.82	$1^{(-)},3^{(-)}$	88.80	43.98	$1^{(+)},2^{(+)}$
unc-s-w	100	2000	100	3128.43	188.36	$2^{(+)},3^{(-)}$	5911.11	534.24	$1^{(-)},3^{(-)}$	2106.45	249.28	$1^{(+)},2^{(+)}$
	100	2000	1000	606.14	99.23	$2^{(+)},3^{(-)}$	1564.23	619.97	$1^{(-)},3^{(-)}$	302.34	24.60	$1^{(+)},2^{(+)}$
	100	2000	5000	147.55	31.80	$3^{(-)}$	174.23	95.98	$3^{(-)}$	60.94	9.12	$1^{(+)},2^{(+)}$
	100	2000	15000	64.65	17.13	$2^{(-)},3^{(-)}$	40.66	15.51	$1^{(+)},3^{(-)}$	19.26	4.04	$1^{(+)},2^{(+)}$
bou-s-c	100	2000	100	3271.07	266.54	$2^{(+)}$	5583.53	337.81	$1^{(-)},3^{(-)}$	3036.97	297.33	$2^{(+)}$
	100	2000	1000	1483.01	85.14	$2^{(+)},3^{(-)}$	2639.16	106.47	$1^{(-)},3^{(-)}$	617.92	186.35	$1^{(+)},2^{(+)}$
	100	2000	5000	796.77	89.80	$2^{(+)},3^{(-)}$	1256.62	118.27	$1^{(-)},3^{(-)}$	251.41	109.58	$1^{(+)},2^{(+)}$
	100	2000	15000	538.45	66.98	$2^{(+)},3^{(-)}$	687.95	116.91	$1^{(-)},3^{(-)}$	104.27	61.06	$1^{(+)},2^{(+)}$
uncor	100	10000	100	10256.72	210.51	$2^{(+)},3^{(+)}$	16278.97	248.43	$1^{(-)},3^{(-)}$	11038.07	236.91	$1^{(-)},2^{(+)}$
	100	10000	1000	3604.18	285.73	$2^{(+)}$	13340.20	704.32	$1^{(-)},3^{(-)}$	3508.51	473.42	$2^{(+)}$
	100	10000	5000	1607.78	278.60	$2^{(+)},3^{(-)}$	10614.45	1660.32	$1^{(-)},3^{(-)}$	1183.52	411.83	$1^{(+)},2^{(+)}$
	100	10000	15000	987.64	219.53	$2^{(+)},3^{(-)}$	8006.35	1612.20	$1^{(-)},3^{(-)}$	566.69	219.54	$1^{(+)},2^{(+)}$
unc-s-w	100	10000	100	7192.82	153.93	$2^{(+)},3^{(+)}$	12617.69	318.23	$1^{(-)},3^{(-)}$	8057.44	274.17	$1^{(-)},2^{(+)}$
	100	10000	1000	1846.43	115.23	$2^{(+)}$	6981.81	768.78	$1^{(-)},3^{(-)}$	1743.12	364.38	$2^{(+)}$
	100	10000	5000	539.39	65.39	$2^{(+)}$	3488.28	819.51	$1^{(-)},3^{(-)}$	519.63	175.22	$2^{(+)}$
	100	10000	15000	208.73	36.91	$2^{(+)}$	1525.23	306.72	$1^{(-)},3^{(-)}$	201.97	79.28	$2^{(+)}$
bou-s-c	100	10000	100	7187.80	122.59	$2^{(+)},3^{(+)}$	15111.38	231.53	$1^{(-)},3^{(-)}$	12736.55	229.48	$1^{(-)},2^{(+)}$
	100	10000	1000	2282.81	219.24	$2^{(+)},3^{(+)}$	8301.43	569.90	$1^{(-)},3^{(-)}$	3575.26	550.54	$1^{(-)},2^{(+)}$
	100	10000	5000	1370.48	250.59	$2^{(+)}$	5248.40	1045.78	$1^{(-)},3^{(-)}$	1472.19	493.88	$2^{(+)}$
	100	10000	15000	955.38	133.33	$2^{(+)}$	3852.07	752.84	$1^{(-)},3^{(-)}$	977.41	397.75	$2^{(+)}$

it can be seen in the Table 7.2, however, there is only one case in which MOEA beat the (1+1) EA: when the weights are similar, and the magnitude of changes are small (2000), which means the population size is also small (in comparison to 10000), and finally τ is at its maximum to let the MOEA to use its population to optimise the problem.

Although MOEA does not perform very well in instances with general weights, the multi-objective approach with a better defined dominance, MOEA_D, does outperform (1+1) EA in many cases. We compare the performance of (1+1) EA and MOEA_D below.

When changes are smaller, it can be seen in Table 7.2 that the mean of offline errors of MOEA_D is smaller than (1+1) EA. The dominance of MOEA_D is such that only keeps the dominant solutions. When a new solution is found, the algorithm compare it to all of the population, removes solutions that are dominated by it and keeps it only if it is not dominated by the any other one. This process improves the quality of the solutions by increasing the probability of keeping a solution beneficial to future generations. Moreover, it reduces the size of the population significantly. Large changes to the capacity, however, makes the MOEA_D keep more individuals, and it is in this circumstance that (1+1) EA may perform better than MOEA_D.

When $r = 10000$, MOEA_D does not have significantly better results in all cases unlike in the case of $r = 2000$, and in most of the situations it performs as well as

TABLE 7.3: The mean, standard deviation values and statistical tests of the offline error for (1+1) EA, MOEA, MOEA_D based on the normal distribution.

	n	σ	τ	(1+1) EA (1)			MOEA (2)			MOEA_D (3)		
				mean	st	stat	mean	st	stat	mean	st	stat
uncor	100	100	100	2714.72	106.06	$2^{(+)},3^{(+)}$	9016.83	2392.48	$1^{(-)},3^{(-)}$	4271.09	789.94	$1^{(-)},2^{(+)}$
	100	100	1000	1386.66	97.11	$2^{(+)},3^{(-)}$	3714.89	737.11	$1^{(-)},3^{(-)}$	412.89	27.25	$1^{(+)},2^{(+)}$
	100	100	5000	801.54	73.67	$2^{(+)},3^{(-)}$	1266.35	119.25	$1^{(-)},3^{(-)}$	108.28	14.22	$1^{(+)},2^{(+)}$
	100	100	15000	549.71	78.98	$2^{(+)},3^{(-)}$	749.86	148.03	$1^{(-)},3^{(-)}$	61.93	17.03	$1^{(+)},2^{(+)}$
unc-s-w	100	100	100	412.24	111.07	$2^{(+)},3^{(+)}$	1979.65	914.35	$1^{(-)}$	1904.09	877.55	$1^{(-)}$
	100	100	1000	85.55	23.13	$2^{(+)},3^{(+)}$	1566.54	409.32	$1^{(-)}$	1482.37	391.75	$1^{(-)}$
	100	100	5000	36.94	13.61	$2^{(+)},3^{(+)}$	1414.66	448.78	$1^{(-)}$	1322.35	414.27	$1^{(-)}$
	100	100	15000	29.14	19.70	$2^{(+)},3^{(+)}$	1237.67	665.27	$1^{(-)}$	1137.80	648.73	$1^{(-)}$
bou-s-c	100	100	100	1491.36	260.72	$2^{(+)},3^{(+)}$	4625.49	1302.52	$1^{(-)},3^{(-)}$	2903.77	717.92	$1^{(-)},2^{(+)}$
	100	100	1000	736.10	53.99	$2^{(+)},3^{(-)}$	1748.61	189.94	$1^{(-)},3^{(-)}$	312.88	35.52	$1^{(+)},2^{(+)}$
	100	100	5000	446.94	39.36	$2^{(+)},3^{(-)}$	640.60	91.29	$1^{(-)},3^{(-)}$	101.21	17.47	$1^{(+)},2^{(+)}$
	100	100	15000	337.85	40.44	$2^{(+)},3^{(-)}$	469.16	93.99	$1^{(-)},3^{(-)}$	70.16	22.26	$1^{(+)},2^{(+)}$
uncor	100	500	100	4013.84	699.56	$2^{(+)},3^{(-)}$	10133.28	1128.57	$1^{(-)},3^{(-)}$	2469.58	649.04	$1^{(+)},2^{(+)}$
	100	500	1000	1991.43	163.25	$2^{(+)},3^{(-)}$	5205.30	635.00	$1^{(-)},3^{(-)}$	511.58	187.21	$1^{(+)},2^{(+)}$
	100	500	5000	1110.36	86.81	$2^{(+)},3^{(-)}$	1965.38	203.34	$1^{(-)},3^{(-)}$	143.28	54.20	$1^{(+)},2^{(+)}$
	100	500	15000	732.32	81.25	$2^{(+)},3^{(-)}$	953.22	125.64	$1^{(-)},3^{(-)}$	45.26	13.87	$1^{(+)},2^{(+)}$
unc-s-w	100	500	100	1686.42	272.35	$2^{(+)},3^{(+)}$	4739.46	1283.37	$1^{(-)},3^{(-)}$	2693.60	580.74	$1^{(-)},2^{(+)}$
	100	500	1000	262.12	57.43	$2^{(+)}$	766.41	438.22	$1^{(-)},3^{(-)}$	304.96	124.57	$2^{(+)}$
	100	500	5000	75.09	16.18	$3^{(-)}$	86.91	42.32	$3^{(-)}$	47.31	14.83	$1^{(+)},2^{(+)}$
	100	500	15000	37.60	10.96	$2^{(-)},3^{(-)}$	28.57	9.70	$1^{(+)},3^{(-)}$	15.82	4.18	$1^{(+)},2^{(+)}$
bou-s-c	100	500	100	2523.48	244.20	$2^{(+)},3^{(-)}$	4778.00	498.80	$1^{(-)},3^{(-)}$	2248.91	85.01	$1^{(+)},2^{(+)}$
	100	500	1000	1075.70	144.73	$2^{(+)},3^{(-)}$	1862.45	236.14	$1^{(-)},3^{(-)}$	343.62	72.49	$1^{(+)},2^{(+)}$
	100	500	5000	579.38	81.32	$2^{(+)},3^{(-)}$	717.55	50.92	$1^{(-)},3^{(-)}$	99.07	42.41	$1^{(+)},2^{(+)}$
	100	500	15000	407.41	53.79	$3^{(-)}$	358.09	44.40	$3^{(-)}$	33.33	13.59	$1^{(+)},2^{(+)}$

(1+1) EA. In all high frequency conditions where $\tau = 100$, the (1+1) EA has better performance. It may be caused by MOEA_D needing more time to optimise a population with a larger size. Moreover, when the magnitude of changes is large, it is more likely that a new change will force MOEA_D to remove all of its stored individuals and start from scratch.

We now study the experimental results that come from considering the dynamic changes under the normal distribution (Table 7.3). The results confirm that (1+1) EA is faster when changes are more frequent. When the variation of changes is small, skipping the case with uncorrelated similar weights and frequent changes, MOEA_D has always been the best algorithm in terms of performance and MOEA has been the worst.

The most notable results occur in the case with uncorrelated similar weights and small σ. (1+1) EA outperforms both other algorithms in this instance. This happens because of the value of δ and the weights of the instances. δ is set to 2σ in the multi-objective approaches and the weights of items are integers in $[1001, 1010]$ in this type of instance. (1+1) EA is able to freely get closer to the optimal solutions from both directions, while the multi-objective approaches are only allowed to consider solutions in range of $[C - \delta, C + \delta]$. In other words, it is possible that there is only one solution in that range or even no solution. Thus, the multi-objective approaches either do not find any feasible solution and get penalty in the offline error, or are not able to improve their feasible solution. Hence, multi-objective approaches have no

advantage in this type of instances according to the value of δ and weights of the items, and in fact, may have a disadvantage.

On the other hand, increasing σ to 500 is enough for MOEA and MOEA_D to benefit from the population again. Although (1+1) EA outperforms them in $\tau = 100$, it is not the absolute dominant algorithm in instances with uncorrelated similar weights anymore. More precisely, the results show that there is no significant difference between their performances in bounded strongly correlated instances and $\tau = 15000$. Furthermore, the use of population in MOEA even cause a significantly better performance in low frequent changes and uncorrelated similar weights instances.

7.4 NSGA-II and SPEA2

The results presented in Section 7.3.2 demonstrate the advantage of solving dynamic KP as a multi-objective optimisation problem. We showed that using populations, specifically in low frequent alterations, improve the efficiency and the quality of founded solutions by preparing the algorithms for the coming dynamic changes. In this section, we analyse the performance of NSGA-II and SPEA2 on the dynamic knapsack problem. Both algorithms are well established approaches in the area of evolutionary multi-objective optimisation. We are interested in analysing the advantages of their heuristic techniques in comparison to each other and also in the environments with high frequent changes. Moreover, we compare their performance with MOEA_D as the best baseline algorithm of the previous section.

As both of these algorithms, unlike MOEA_D, use specific techniques to guarantee a well-distributed solution set, to consider the capacity constraint and also prepare the algorithm for the following dynamic changes, we present a new formulation of the problem. Moreover, we discuss the elitism in each of the algorithms and show how the elitism in plain versions, which is in favour of more distributed solutions, causes the loss of good quality solution with respect to the profit and capacity constraint. Next, we apply an additional elitism to improve the performance against the plain versions of the algorithms.

7.4.1 New formulation for Dynamic KP

In this section, we present a new fitness evaluation approach, different from the one used for MOEA_D, for NSGA-II and SPEA2 to solve the dynamic knapsack problem. In contrast to MOEA_D which uses two separate solution sets to store infeasible solutions to prepare for the following dynamic changes, we benefit from the ability of SPEA2 and NSGA-II in finding a well-distributed non-dominated set. We force the algorithms to find non-dominated solutions with weights within the interval $[C - \delta, C + \delta]$. To this aim, we apply penalty on the weights and profits of

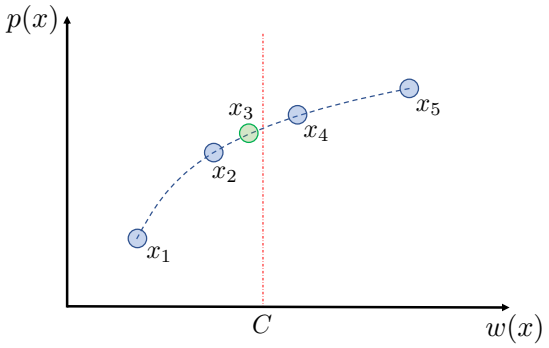


FIGURE 7.2: A situation that NSGA-II and SPEA2 lose the best feasible solution because of the impact of distance factor on the solution ranking.

solutions outside of the interval such that for any solution x we have

$$w_{MO}(x) = \begin{cases} w(x) & \text{if } w(x) \in [C-\delta, C+\delta] \\ w(x) + (n \cdot w_{max} + 1) \cdot \alpha(x) & \text{otherwise,} \end{cases} \tag{7.1}$$

where for solution x that $w(x) \notin [C-\delta, C+\delta]$, $\alpha(x) = \min\{|w(x) - C - \delta|, |w(x) - C + \delta|\}$ is the distance from the edge of the interval. Similar to the weight, we apply penalty on the profit as follows

$$p_{MO}(x) = \begin{cases} p(x) & \text{if } w(x) \in [C-\delta, C+\delta] \\ p(x) - (n \cdot p_{max} + 1) \cdot \alpha(x) & \text{otherwise.} \end{cases} \tag{7.2}$$

Note that the objectives are weight $\big(w_{MO}(x)\big)$ and profit $\big(p_{MO}(x)\big)$ which should be minimised and maximised, respectively. The penalty guarantees that any solution in the preferred interval dominate the solutions outside and solutions that are closer to the interval dominate farther ones. In this way if the algorithms produce a well-distributed non-dominated solutions, we expect to have good quality feasible solutions even after the dynamic change.

7.4.2 Additional Elitism

While we still look for the feasible solution with the highest profit, the new formulation does not apply penalties on all infeasible solutions and there are better solutions in the population in terms of profit value which their weight exceed the capacity constraint. In NSGA-II and SPEA2, solutions are preferred according to their rank, which is calculated based on the number of solutions that dominate them, and their distance from the other solutions in the objective space. Although the best feasible solution is a non-dominated solution, it is possible that the algorithms lose it because of the distance factor. Figure 7.2 demonstrates such a situation in which the algorithms have to pick 4 solutions from non-dominated set $\{x_1, \cdots, x_5\}$ to produce the next generation. Note that solutions x_1, x_2 and x_3 are feasible, and the others

are infeasible. Both of the algorithms tend to keep x_1 and x_5 since they are border solutions and have the maximum distance factor. Thus, they have to choose 2 solutions within $\{x_2, x_3, x_4\}$. Although, x_3 is the best feasible solution in this situation, the algorithms pick x_2 and x_4 since they provide a better distributed set. To address this problem, we artificially change the distance factor of the best feasible solution so that the algorithms keep it in the next generation.

To apply it on the NSGA-II, we store the best feasible solution in a separate variable. After line 10, we check if a new feasible solution dominated the previous one. If the stored best solution has been removed from the population set after line 10, and the best feasible solution in P_{t+1} has less profit value, we artificially remove the worst solution from P_{t+1} with regards to the front ranks and crowding distances, add the stored best solution to the first front of P_{t+1}, and assign infinity value to its crowding distance. Otherwise, either the best feasible solution is still in P_{t+1} or the algorithm has found a better feasible solution. Hence, we only update the stored best feasible solution and assign the infinity value to its crowding distance. Note that by performing this approach, we assign more reproduction power the best feasible solution since it is always the winner of selection phase. Hence, it is more probable that we update the feasible solution close to the constraint and achieve a better feasible solution.

In SPEA2, the elitism procedure is similar to NSGA-II. We store the best feasible solution and either update it or artificially add it to archive set in each generation after environmental selection (Line 6 in Algorithm 6). However, to make sure that it has a higher probability of reproduction, we assign zero to its fitness value. In this case, the solution is the outcome of tournament selection phase and has more opportunity to produce an offspring.

Note that changing the fitness values happens exactly before the reproduction steps. Since both algorithms re-evaluate fitness values prior to selecting the parents, the elitism approach guarantees that if another solution dominate the current best feasible solution then we remove it from the population. Hence, it does not affect the natural behaviour of the algorithms.

7.4.3 Experimental Results

We compare NSGA-II and SPEA2 results with MOEA_D as the previous winner algorithm in most of the cases. While the definition of dominance is the only limitation on the population size in MOEA_D, we set the population size for NSGA-II and SPEA2 to 20.

In addition to the total offline error, introduced in Section 7.2.2, we also compare the algorithms based on a new factor, called partial offline error. This factor considers the best feasible solution achieved by an algorithm right before a dynamic change, i.e., the performance of algorithms are analysed based on their final population only.

This factor illustrates the final achievement of the algorithms during a "no change" period and does not consider the performance of the algorithms within the optimisation process. To compute the best partial offline error, instead of all the generations, we record the profit value of best feasible solution only in the last generation before the next change. Let x_i denote the best feasible solution before $i+1$th change happens and x_i^* denote the optimal solution corresponding to the ith capacity. If there is no feasible solution in the last generation, y_i denotes the solution with lowest $v(x)$, the constraint violation of solution x. Then, the partial offline error is calculated as follow:

$$E_{BO} = \sum_{i=1}^{\lfloor 10^6/\tau \rfloor} \frac{e_i}{\lfloor 10^6/\tau \rfloor}, \tag{7.3}$$

where $e(x)$ if the offline error of solution x calculated as follows:

$$e_i = \begin{cases} p(x_i^*) - p(x_i) & \text{if } v(x_i) = 0 \\ p(x_i^*) + v(y_i) & \text{otherwise.} \end{cases} \tag{7.4}$$

7.4.4 Analysis

We now present a detailed analysis on the performance of simple and advanced baseline evolutionary algorithms on the dynamic knapsack problem in our experiments.

In Table 7.4, we compare the different algorithms: MOEA_D, NSGA-II, SPEA2 and two novel algorithms NSGA-II(we) and SPEA2(we) based on the elitism mechanism. We summarise our results in terms of the mean value of the offline error achieved for each instance based on the uniform distribution.

The displayed results show NSGA-II and SPEA2 significantly outperform MOEA_D when the changes occur more frequently, i.e., $\tau = 100, 1000$ for most considered instances. However, MOEA_D achieves better results for the uncorrelated similar weights instances compared to NSGA-II and SPEA2 when the frequently is low. For the frequency $\tau = 15000$, the MOEA_D algorithm especially has sufficient time to find a good solution. When comparing MOEA_D, NSGA-II, SPEA2 with algorithms based on elitism, NSGA-II(we) and SPEA2(we) achieve the best results among all different instances in most cases. This may indicate that both algorithms effectively used the elitism strategy outlined in Section 7.4.2.

In Table 7.5, we compare these algorithms for different instances based on the Normal distribution. We investigate the combinations; $\sigma = 100$ and $\sigma = 500$, and $\tau = 100, 1000, 50000, 150000$. MOEA_D is usually outperformed by SPEA2. However, with the exception of the following cases: uncorrelated similar weights, $\sigma = 100$, $\tau = 5000, 1500$, and uncorrelated, uncorrelated similar weights, $\sigma = 500$, $\tau = 5000, 1500$, MOEA_D performed better. Furthermore, by comparing MOEA_D and NSGA-II,

TABLE 7.4: The mean, standard deviation values and statistical tests of the total offline error for MOEA_D, NSGA-II, SPEA2, NSGA-II with elitism and, SPEA2 with elitism based on the uniform distribution.

	n	r	τ	MOEA_D (1)			NSGA-II (2)			SPEA2 (3)			NSGA-II(we) (4)			SPEA2(we) (5)		
				mean	st	stat	mean	st	stat	mean	st	stat	mean	st	stat	mean	st	stat
uncor	100	2000	100	3684.26	525.50	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	134.29	30.53	$1^{(+)},4^{(-)},5^{(-)}$	123.12	31.65	$1^{(+)},4^{(-)},5^{(-)}$	48.72	14.43	$1^{(+)},2^{(+)},3^{(+)}$	50.74	18.50	$1^{(+)},2^{(+)},3^{(+)}$
	100	2000	1000	776.14	334.69	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	126.57	66.70	$1^{(+)},4^{(-)},5^{(-)}$	103.17	57.02	$1^{(+)},4^{(-)},5^{(-)}$	8.05	4.76	$1^{(+)},2^{(+)},3^{(+)}$	6.51	5.40	$1^{(+)},2^{(+)},3^{(+)}$
	100	2000	5000	270.90	121.44	$4^{(-)},5^{(-)}$	168.85	112.04	$4^{(-)},5^{(-)}$	137.66	92.84	$4^{(-)},5^{(-)}$	3.05	2.17	$1^{(+)},2^{(+)},3^{(+)}$	2.33	3.26	$1^{(+)},2^{(+)},3^{(+)}$
	100	2000	15000	88.80	43.98	$4^{(-)},5^{(-)}$	140.99	141.45	$4^{(-)},5^{(-)}$	115.72	116.15	$4^{(-)},5^{(-)}$	1.04	1.20	$1^{(+)},2^{(+)},3^{(+)}$	0.62	1.03	$1^{(+)},2^{(+)},3^{(+)}$
unc-s-w	100	2000	100	2106.45	249.28	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	10.35	2.43	$1^{(+)},3^{(+)},5^{(+)}$	134.24	40.80	$1^{(+)},2^{(-)},4^{(-)}$	8.71	1.92	$1^{(+)},3^{(+)},5^{(+)}$	64.11	20.40	$1^{(+)},2^{(-)},4^{(-)}$
	100	2000	1000	302.34	24.60	$2^{(-)},4^{(-)},5^{(-)}$	3.37	2.35	$1^{(+)},3^{(+)},5^{(+)}$	209.79	96.26	$2^{(-)},4^{(-)},5^{(-)}$	1.18	0.51	$1^{(+)},3^{(+)},5^{(+)}$	24.76	15.37	$1^{(+)},2^{(-)},3^{(+)},4^{(-)}$
	100	2000	5000	60.94	9.12	$2^{(-)},4^{(-)},5^{(-)}$	3.78	3.02	$1^{(+)},3^{(+)}$	357.82	125.60	$2^{(-)},4^{(-)},5^{(-)}$	0.56	0.39	$1^{(+)},3^{(+)},5^{(+)}$	13.42	5.70	$1^{(+)},3^{(+)},4^{(-)}$
	100	2000	15000	19.26	4.04	$2^{(-)},4^{(-)},5^{(-)}$	4.08	4.43	$1^{(+)},3^{(+)}$	353.30	166.59	$2^{(-)},4^{(-)},5^{(-)}$	0.36	0.47	$1^{(+)},3^{(+)},5^{(+)}$	5.29	4.21	$1^{(+)},3^{(+)},4^{(-)}$
bou-s-c	100	2000	100	3036.97	297.34	$3^{(-)},4^{(-)},5^{(-)}$	224.80	6.82	$4^{(-)},5^{(-)}$	203.96	6.62	$1^{(+)},5^{(-)}$	71.96	2.46	$1^{(+)},2^{(+)}$	58.13	4.74	$1^{(+)},2^{(+)},3^{(+)}$
	100	2000	1000	617.92	186.35	$3^{(-)},4^{(-)},5^{(-)}$	235.49	15.10	$4^{(-)},5^{(-)}$	200.86	9.40	$1^{(+)},5^{(-)}$	24.82	1.96	$1^{(+)},2^{(+)}$	15.36	1.61	$1^{(+)},2^{(+)},3^{(+)}$
	100	2000	5000	251.41	109.58	$4^{(-)},5^{(-)}$	242.63	17.76	$3^{(-)},4^{(-)},5^{(-)}$	204.60	7.40	$2^{(+)},4^{(-)},5^{(-)}$	10.36	2.44	$1^{(+)},2^{(+)},3^{(+)}$	4.76	1.21	$1^{(+)},2^{(+)},3^{(+)}$
	100	2000	15000	104.27	61.06	$2^{(+)},4^{(-)},5^{(-)}$	244.34	25.98	$1^{(-)},4^{(-)},5^{(-)}$	207.38	12.74	$4^{(-)},5^{(-)}$	5.74	2.27	$1^{(+)},2^{(+)},3^{(+)}$	2.07	1.08	$1^{(+)},2^{(+)},3^{(+)}$
uncor	100	10000	100	11038.07	236.91	$2^{(-)},4^{(-)},5^{(-)}$	1227.93	62.11	$1^{(+)},3^{(+)},4^{(-)}$	1688.41	95.98	$2^{(-)},4^{(-)},5^{(-)}$	944.55	57.23	$1^{(+)},2^{(+)},3^{(+)},5^{(+)}$	1387.19	92.72	$1^{(+)},3^{(+)},4^{(-)}$
	100	10000	1000	3508.51	473.42	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	898.80	126.18	$1^{(+)},4^{(-)},5^{(-)}$	1083.25	203.47	$1^{(+)},4^{(-)},5^{(-)}$	197.58	34.04	$1^{(+)},2^{(+)},3^{(+)}$	380.65	94.12	$1^{(+)},2^{(+)},3^{(+)}$
	100	10000	5000	1183.52	411.83	$4^{(-)},5^{(-)}$	951.97	256.55	$4^{(-)},5^{(-)}$	987.39	351.94	$4^{(-)},5^{(-)}$	99.76	32.32	$1^{(+)},2^{(+)},3^{(+)}$	106.14	50.70	$1^{(+)},2^{(+)},3^{(+)}$
	100	10000	15000	566.70	219.54	$4^{(-)},5^{(-)}$	953.69	459.12	$4^{(-)},5^{(-)}$	927.86	613.20	$4^{(-)},5^{(-)}$	59.96	35.22	$1^{(+)},2^{(+)},3^{(+)}$	40.05	32.27	$1^{(+)},2^{(+)},3^{(+)}$
unc-s-w	100	10000	100	8057.44	274.17	$2^{(-)},4^{(-)},5^{(-)}$	562.54	34.34	$1^{(+)},4^{(-)}$	653.70	41.75	$4^{(-)},5^{(-)}$	283.83	18.20	$1^{(+)},2^{(+)},3^{(+)}$	396.46	34.89	$1^{(+)},3^{(+)}$
	100	10000	1000	1743.12	364.38	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	601.19	108.71	$1^{(+)},4^{(-)},5^{(-)}$	584.00	106.60	$1^{(+)},4^{(-)},5^{(-)}$	116.02	26.00	$1^{(+)},2^{(+)},3^{(+)}$	107.31	28.42	$1^{(+)},2^{(+)},3^{(+)}$
	100	10000	5000	519.63	175.22	$4^{(-)},5^{(-)}$	638.72	224.53	$4^{(-)},5^{(-)}$	593.92	226.11	$4^{(-)},5^{(-)}$	58.83	30.19	$1^{(+)},2^{(+)},3^{(+)}$	41.63	24.35	$1^{(+)},2^{(+)},3^{(+)}$
	100	10000	15000	201.97	79.28	$2^{(+)},3^{(+)},4^{(-)},5^{(-)}$	618.73	277.40	$1^{(-)},4^{(-)},5^{(-)}$	549.70	266.72	$1^{(-)},4^{(-)},5^{(-)}$	25.41	19.56	$1^{(+)},2^{(+)},3^{(+)}$	15.05	13.88	$1^{(+)},2^{(+)},3^{(+)}$
bou-s-c	100	10000	100	12736.55	229.48	$2^{(-)},4^{(-)},5^{(-)}$	1449.09	39.45	$1^{(+)},4^{(-)}$	1625.39	64.99	$4^{(-)},5^{(-)}$	693.73	38.56	$1^{(+)},2^{(+)},3^{(+)}$	878.51	62.47	$1^{(+)},3^{(+)}$
	100	10000	1000	3575.26	550.54	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	1421.59	105.78	$1^{(+)},4^{(-)},5^{(-)}$	1515.04	145.19	$1^{(+)},4^{(-)},5^{(-)}$	306.22	38.62	$1^{(+)},2^{(+)},3^{(+)}$	386.50	61.47	$1^{(+)},2^{(+)},3^{(+)}$
	100	10000	5000	1472.19	493.88	$4^{(-)},5^{(-)}$	1419.66	223.08	$4^{(-)},5^{(-)}$	1491.40	281.66	$4^{(-)},5^{(-)}$	176.12	47.55	$1^{(+)},2^{(+)},3^{(+)}$	202.88	48.35	$1^{(+)},2^{(+)},3^{(+)}$
	100	10000	15000	977.42	397.75	$4^{(-)},5^{(-)}$	1349.77	294.06	$4^{(-)},5^{(-)}$	1448.12	369.69	$4^{(-)},5^{(-)}$	109.48	40.90	$1^{(+)},2^{(+)},3^{(+)}$	129.12	41.19	$1^{(+)},2^{(+)},3^{(+)}$

TABLE 7.5: The mean, standard deviation values and statistical tests of the total offline error for MOEA_D, NSGA-II, SPEA2, NSGA-II with elitism, and SPEA2 with elitism based on the normal distribution.

	n	σ	τ	MOEA_D (1)			NSGA-II (2)			SPEA2 (3)			NSGA-II(we) (4)			SPEA2(we) (5)		
				mean	st	stat	mean	st	stat	mean	st	stat	mean	st	stat	mean	st	stat
uncor	100	100	100	4271.09	789.94	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	9.67	1.85	$1^{(+)},3^{(-)},5^{(-)}$	6.66	2.25	$1^{(+)},2^{(+)}$	7.34	2.08	$1^{(+)}$	5.48	2.59	$1^{(+)},2^{(+)}$
	100	100	1000	412.89	27.25	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	3.79	1.89	$1^{(+)},3^{(-)},4^{(-)},5^{(-)}$	1.61	1.14	$1^{(+)},2^{(+)},5^{(-)}$	1.08	0.45	$1^{(+)},2^{(+)}$	0.58	0.31	$1^{(+)},2^{(+)},3^{(+)}$
	100	100	5000	108.29	14.22	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	5.03	3.59	$1^{(+)},4^{(-)},5^{(-)}$	2.82	2.95	$1^{(+)},5^{(-)}$	0.44	0.34	$1^{(+)},2^{(+)}$	0.11	0.10	$1^{(+)},2^{(+)},3^{(+)}$
	100	100	15000	61.93	17.03	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	3.79	2.79	$1^{(+)},4^{(-)},5^{(-)}$	1.82	2.12	$1^{(+)},5^{(-)}$	0.18	0.18	$1^{(+)},2^{(+)}$	0.05	0.08	$1^{(+)},2^{(+)},3^{(+)}$
unc-s-w	100	100	100	1904.09	877.55	$2^{(+)},3^{(+)}$	7875.64	4651.35	$1^{(-)},4^{(-)},5^{(-)}$	7755.18	4609.32	$1^{(-)},4^{(-)},5^{(-)}$	3905.34	2496.65	$2^{(+)},3^{(+)}$	3526.70	2150.96	$2^{(+)},3^{(+)}$
	100	100	1000	1482.37	391.75	$2^{(+)},3^{(+)},4^{(+)},5^{(+)}$	8914.58	1233.57	$1^{(-)},4^{(-)},5^{(-)}$	8981.25	1260.77	$1^{(-)},4^{(-)},5^{(-)}$	4264.04	1011.79	$1^{(-)},2^{(+)},3^{(+)}$	4169.18	992.44	$1^{(-)},2^{(+)},3^{(+)}$
	100	100	5000	1322.35	414.28	$2^{(+)},3^{(+)}$	4490.67	1477.30	$1^{(-)},4^{(-)},5^{(-)}$	4502.98	1486.96	$1^{(-)},4^{(-)},5^{(-)}$	2102.75	1241.46	$2^{(+)},3^{(+)}$	2135.04	1202.39	$2^{(+)},3^{(+)}$
	100	100	15000	1137.80	648.73	$2^{(+)},3^{(+)}$	4585.86	1141.78	$1^{(-)},4^{(-)},5^{(-)}$	4598.89	1147.26	$1^{(-)},4^{(-)},5^{(-)}$	1863.60	1321.14	$2^{(+)},3^{(+)}$	1954.11	1455.11	$2^{(+)},3^{(+)}$
bou-s-c	100	100	100	2903.77	717.92	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	20.47	3.55	$1^{(+)},3^{(-)},4^{(-)},5^{(-)}$	14.45	2.00	$1^{(+)},2^{(+)},5^{(-)}$	11.99	2.80	$1^{(+)},2^{(+)}$	9.80	1.97	$1^{(+)},2^{(+)},3^{(+)}$
	100	100	1000	312.88	35.53	$3^{(-)},4^{(-)},5^{(-)}$	17.22	0.73	$4^{(-)},5^{(-)}$	12.32	1.10	$1^{(+)},4^{(-)},5^{(-)}$	3.89	0.60	$1^{(+)},2^{(+)},3^{(+)}$	3.61	1.15	$1^{(+)},2^{(+)},3^{(+)}$
	100	100	5000	101.21	17.47	$3^{(-)},4^{(-)},5^{(-)}$	17.02	0.84	$4^{(-)},5^{(-)}$	12.62	1.97	$1^{(+)},4^{(-)},5^{(-)}$	1.44	0.30	$1^{(+)},2^{(+)},3^{(+)}$	1.10	0.40	$1^{(+)},2^{(+)},3^{(+)}$
	100	100	15000	70.16	22.26	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	17.04	0.85	$1^{(+)},4^{(-)},5^{(-)}$	13.37	2.13	$1^{(+)},4^{(-)},5^{(-)}$	0.73	0.25	$1^{(+)},2^{(+)},3^{(+)}$	0.46	0.38	$1^{(+)},2^{(+)},3^{(+)}$
uncor	100	500	100	2469.58	649.04	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	54.72	30.38	$1^{(+)},4^{(-)},5^{(-)}$	41.16	24.57	$1^{(+)},4^{(-)},5^{(-)}$	16.93	9.82	$1^{(+)},2^{(+)},3^{(+)}$	13.81	9.04	$1^{(+)},2^{(+)},3^{(+)}$
	100	500	1000 "	511.58	187.21	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	70.79	52.31	$1^{(+)},4^{(-)},5^{(-)}$	51.23	39.45	$1^{(+)},4^{(-)},5^{(-)}$	3.75	2.51	$1^{(+)},2^{(+)},3^{(+)}$	2.47	1.87	$1^{(+)},2^{(+)},3^{(+)}$
	100	500	5000	143.28	54.20	$4^{(-)},5^{(-)}$	108.83	57.57	$4^{(-)},5^{(-)}$	80.00	45.64	$4^{(-)},5^{(-)}$	1.35	0.45	$1^{(+)},2^{(+)},3^{(+)}$	0.88	0.63	$1^{(+)},2^{(+)},3^{(+)}$
	100	500	15000	45.26	13.87	$2^{(+)},3^{(+)},4^{(-)},5^{(-)}$	142.80	60.99	$1^{(-)},4^{(-)},5^{(-)}$	107.71	48.50	$1^{(-)},4^{(-)},5^{(-)}$	0.58	0.18	$1^{(+)},2^{(+)},3^{(+)}$	0.44	0.38	$1^{(+)},2^{(+)},3^{(+)}$
unc-s-w	100	500	100	2693.60	580.74	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	29.12	8.72	$1^{(+)},3^{(+)},5^{(+)}$	164.38	61.26	$1^{(+)},2^{(-)},4^{(-)}$	28.72	8.57	$1^{(+)},3^{(+)},5^{(+)}$	78.35	29.19	$1^{(+)},2^{(-)},4^{(-)}$
	100	500	1000	304.96	124.57	$2^{(-)},4^{(-)},5^{(-)}$	6.92	2.73	$1^{(+)},3^{(+)},5^{(+)}$	413.96	221.74	$2^{(-)},4^{(-)},5^{(-)}$	5.93	3.00	$1^{(+)},3^{(+)},5^{(+)}$	52.51	22.42	$1^{(+)},2^{(-)},3^{(+)},4^{(-)}$
	100	500	5000	47.31	14.83	$2^{(-)},4^{(-)},5^{(-)}$	2.65	1.51	$1^{(+)},3^{(+)}$	402.22	328.16	$2^{(-)},4^{(-)},5^{(-)}$	1.10	1.10	$1^{(+)},3^{(+)},5^{(+)}$	20.05	16.55	$1^{(+)},3^{(+)},4^{(-)}$
	100	500	15000	15.82	4.18	$2^{(-)},4^{(-)},5^{(-)}$	2.26	2.02	$1^{(+)},3^{(+)}$	264.22	280.33	$2^{(-)},4^{(-)},5^{(-)}$	0.33	0.37	$1^{(+)},3^{(+)},5^{(+)}$	5.62	6.89	$1^{(+)},3^{(+)},4^{(-)}$
bou-s-c	100	500	100	2248.91	85.01	$3^{(-)},4^{(-)},5^{(-)}$	98.45	4.84	$4^{(-)},5^{(-)}$	80.20	3.81	$1^{(+)},5^{(-)}$	32.91	0.98	$1^{(+)},2^{(+)}$	23.64	1.43	$1^{(+)},2^{(+)},3^{(+)}$
	100	500	1000	343.62	72.49	$3^{(-)},4^{(-)},5^{(-)}$	106.57	9.43	$4^{(-)},5^{(-)}$	81.93	6.29	$1^{(+)},5^{(-)}$	9.59	1.48	$1^{(+)},2^{(+)}$	5.22	1.50	$1^{(+)},2^{(+)},3^{(+)}$
	100	500	5000	99.07	42.41	$4^{(-)},5^{(-)}$	112.97	10.10	$3^{(-)},4^{(-)},5^{(-)}$	85.87	8.79	$2^{(+)},4^{(-)},5^{(-)}$	2.82	0.61	$1^{(+)},2^{(+)},3^{(+)}$	1.02	0.26	$1^{(+)},2^{(+)},3^{(+)}$
	100	500	15000	33.33	13.59	$2^{(+)},5^{(-)}$	117.62	11.47	$1^{(-)},4^{(-)},5^{(-)}$	90.26	10.34	$4^{(-)},5^{(-)}$	1.10	0.24	$2^{(+)},3^{(+)}$	0.30	0.08	$1^{(+)},2^{(+)},3^{(+)}$

we observe the same behaviour except for the mean values for the instance bounded strongly correlated where $\sigma = 500$, $\tau = 1500$ are substantially worse than those of MOEA_D. The results show that NSGA-II(we) and SPEA2(we) significantly outperform MOEA_D, NSGA-II and SPEA2 in most of the cases as expected due to the elitism approach.

Table 7.6 clearly indicates that the elitism mechanisms used in NSGA-II(we) and SPEA2(we) achieve substantially better results with respect to the partial offline error compared NSGA-II, SPEA2, and MOEA_D in all instances. Interestingly, NSGA-II(we) benefits much more from elitism than SPEA2(we), and NSGA-II(we) achieves the lowest partial offline error across most instances. An exception are bounded strongly correlated instances with $r = 100$. The MOEA_D algorithm, especially for $r = 2000, 10000$ and $\tau = 15000$ significantly outperforms NSGA-II and SPEA2 by one order of magnitude in terms of offline error in most cases.

Furthermore, Table 7.7 shows results for the the Normal distribution. The results summarise the partial offline error and statistical tests for all five algorithms. NSGA-II(we) and SPEA2(we) perform significantly better than MOEA_D, NSGA-II and SPEA2 in all instances. Noticeably, SPEA2(we) for uncorrelated and bounded strongly correlated instances, $\sigma = 500$, and $\tau = 5000, 15000$ finds the results close to the optimum. Similar to the experiments based on the total offline error, MOEA_D is outperformed by NSGA-II and SPEA2, whereas MOEA_D outperforms NSGA-II and SPEA2 in the instance with uncorrelated similar weights and $\sigma = 100$.

The impact of elitism in MOEA_D, NSGA-II(we), and SPEA2(we) is also illustrated by observing the mean values of the same column in one specific instance. Note that in an environment with low frequent changes the algorithms have more time to find a good solution before the next change happens, i.e., the mean of total/partial offline error for one specific algorithm against the same instance supposed to decrease by increasing the value of τ. However, such pattern could not be seen for NSGA-II and SPEA2, mostly in $\tau \geq 1000$. It shows that NSGA-II and SPEA2 find and fix a well distributed set of solution in the first 1000 iterations, according to the classic distribution techniques, and do not improve the best found solution, which cause adding the same amount of error to the total offline error in the next iterations.

Comparing the results of algorithms in uniform instances according the total and offline partial errors (Tables 7.4,7.6), also illustrates the the importance of customised elitism in advanced evolutionary algorithm. The outperformance of MOEA_D in low frequent changes according to the partial offline error shows that at the end of each interval, it has found a better solution than classic NSGA-II and SPEA2. However, according to the total offline error, this progress is slow and not significantly better in average.

Overall, our results suggest that NSGA-II(we) and SPEA2(we) using the elitism mechanism significantly outperform the classical NSGA-II, SPEA2 and MOEA_D.

TABLE 7.6: The mean, standard deviation values and statistical tests of the partial offline error for MOEA_D, NSGA-II, SPEA2, NSGA-II with elitism and, SPEA2 with elitism based on the uniform distribution.

	n	r	τ	MOEA_D (1)			NSGA-II (2)			SPEA2 (3)			NSGA-II(we) (4)			SPEA2(we) (5)		
				mean	st	stat	mean	st	stat	mean	st	stat	mean	st	stat	mean	st	stat
uncor	100	2000	100	2134.03	377.28	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	135.95	29.93	$1^{(+)},4^{(-)},5^{(-)}$	119.59	29.48	$1^{(+)},4^{(-)},5^{(-)}$	26.04	8.48	$1^{(+)},2^{(+)},3^{(+)}$	25.86	10.33	$1^{(+)},2^{(+)},3^{(+)}$
	100	2000	1000	336.71	179.08	$3^{(-)},4^{(-)},5^{(-)}$	128.84	69.21	$4^{(-)},5^{(-)}$	103.68	57.17	$1^{(+)},4^{(-)},5^{(-)}$	2.31	1.49	$1^{(+)},2^{(+)},3^{(+)}$	0.69	0.75	$1^{(+)},2^{(+)},3^{(+)}$
	100	2000	5000	79.80	47.49	$4^{(-)},5^{(-)}$	168.92	116.40	$4^{(-)},5^{(-)}$	141.67	99.47	$4^{(-)},5^{(-)}$	0.50	0.47	$1^{(+)},2^{(+)},3^{(+)}$	0.01	0.01	$1^{(+)},2^{(+)},3^{(+)}$
	100	2000	15000	14.70	16.17	$2^{(+)},3^{(+)},4^{(-)},5^{(-)}$	141.09	141.93	$1^{(-)},4^{(-)},5^{(-)}$	123.87	128.21	$1^{(-)},4^{(-)},5^{(-)}$	0.13	0.29	$1^{(+)},2^{(+)},3^{(+)}$	0.00	0.00	$1^{(+)},2^{(+)},3^{(+)}$
unc-s-w	100	2000	100	1102.09	113.57	$2^{(-)},4^{(-)},5^{(-)}$	5.71	1.74	$1^{(+)},3^{(+)},5^{(+)}$	133.25	40.74	$2^{(-)},4^{(-)}$	3.59	0.90	$1^{(+)},3^{(+)},5^{(+)}$	54.73	18.74	$1^{(+)},2^{(-)},4^{(-)}$
	100	2000	1000	103.15	15.95	$2^{(-)},4^{(-)},5^{(-)}$	2.55	2.12	$1^{(+)},3^{(+)}$	233.01	101.48	$2^{(-)},4^{(-)},5^{(-)}$	0.27	0.27	$1^{(+)},3^{(+)},5^{(+)}$	17.39	11.62	$1^{(+)},3^{(+)},4^{(-)}$
	100	2000	5000	10.32	5.06	$2^{(-)},3^{(+)},4^{(-)}$	3.49	3.41	$1^{(+)},3^{(+)},4^{(-)}$	391.01	141.59	$1^{(-)},2^{(-)},4^{(-)},5^{(-)}$	0.25	0.27	$1^{(+)},2^{(+)},3^{(+)},5^{(+)}$	6.96	3.67	$3^{(+)},4^{(-)}$
	100	2000	15000	1.34	1.05	$3^{(+)},4^{(-)}$	4.27	4.85	$3^{(+)},4^{(-)}$	381.02	187.18	$1^{(-)},2^{(-)},4^{(-)},5^{(-)}$	0.21	0.41	$1^{(+)},2^{(+)},3^{(+)},5^{(+)}$	1.60	1.94	$3^{(+)},4^{(-)}$
bou-s-c	100	2000	100	1296.54	241.04	$3^{(-)},4^{(-)},5^{(-)}$	230.77	7.05	$4^{(-)},5^{(-)}$	204.91	5.07	$1^{(+)},5^{(-)}$	43.92	1.13	$1^{(+)},2^{(+)}$	33.14	1.81	$1^{(+)},2^{(+)},3^{(+)}$
	100	2000	1000	259.02	92.31	$4^{(-)},5^{(-)}$	236.93	16.89	$4^{(-)},5^{(-)}$	201.48	9.66	$4^{(-)},5^{(-)}$	12.84	1.72	$1^{(+)},2^{(+)},3^{(+)}$	6.00	1.86	$1^{(+)},2^{(+)},3^{(+)}$
	100	2000	5000	96.79	61.33	$2^{(+)},4^{(-)},5^{(-)}$	241.60	23.83	$1^{(-)},4^{(-)},5^{(-)}$	204.39	11.15	$4^{(-)},5^{(-)}$	4.84	2.22	$1^{(+)},2^{(+)},3^{(+)}$	1.01	0.91	$1^{(+)},2^{(+)},3^{(+)}$
	100	2000	15000	37.60	33.88	$2^{(+)},3^{(+)},5^{(-)}$	254.73	34.35	$1^{(-)},4^{(-)},5^{(-)}$	214.77	19.64	$1^{(-)},4^{(-)},5^{(-)}$	2.58	1.82	$2^{(+)},3^{(+)}$	0.23	0.41	$1^{(+)},2^{(+)},3^{(+)}$
uncor	100	10000	100	9410.13	230.91	$2^{(-)},4^{(-)},5^{(-)}$	1084.42	58.24	$1^{(+)},3^{(+)},4^{(-)}$	1448.55	87.95	$2^{(-)},4^{(-)},5^{(-)}$	679.17	47.17	$1^{(+)},2^{(+)},3^{(+)},5^{(+)}$	1001.91	77.73	$1^{(+)},3^{(+)},4^{(-)}$
	100	10000	1000	2345.99	383.96	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	965.88	135.52	$1^{(+)},4^{(-)},5^{(-)}$	995.26	165.62	$1^{(+)},4^{(-)},5^{(-)}$	130.83	22.98	$1^{(+)},2^{(+)},3^{(+)}$	130.05	38.08	$1^{(+)},2^{(+)},3^{(+)}$
	100	10000	5000	698.49	264.02	$4^{(-)},5^{(-)}$	979.27	271.39	$4^{(-)},5^{(-)}$	943.22	345.44	$4^{(-)},5^{(-)}$	58.90	20.05	$1^{(+)},2^{(+)},3^{(+)}$	16.52	7.66	$1^{(+)},2^{(+)},3^{(+)}$
	100	10000	15000	299.81	156.20	$2^{(+)},3^{(+)},4^{(-)},5^{(-)}$	977.11	467.85	$1^{(-)},4^{(-)},5^{(-)}$	956.92	659.17	$1^{(-)},4^{(-)},5^{(-)}$	32.70	20.88	$1^{(+)},2^{(+)},3^{(+)}$	5.21	4.16	$1^{(+)},2^{(+)},3^{(+)}$
unc-s-w	100	10000	100	5981.93	290.09	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	556.04	33.99	$1^{(+)},4^{(-)},5^{(-)}$	614.58	37.51	$1^{(+)},4^{(-)},5^{(-)}$	196.66	13.17	$1^{(+)},2^{(+)},3^{(+)}$	256.44	25.68	$1^{(+)},2^{(+)},3^{(+)}$
	100	10000	1000	998.81	250.14	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	631.46	109.27	$1^{(+)},4^{(-)},5^{(-)}$	591.64	109.36	$1^{(+)},4^{(-)},5^{(-)}$	81.59	20.33	$1^{(+)},2^{(+)},3^{(+)}$	54.06	16.60	$1^{(+)},2^{(+)},3^{(+)}$
	100	10000	5000	238.21	88.11	$2^{(+)},3^{(+)},4^{(-)},5^{(-)}$	662.77	229.14	$1^{(-)},4^{(-)},5^{(-)}$	597.54	229.58	$1^{(-)},4^{(-)},5^{(-)}$	34.91	20.64	$1^{(+)},2^{(+)},3^{(+)}$	16.44	11.31	$1^{(+)},2^{(+)},3^{(+)}$
	100	10000	15000	69.81	33.58	$2^{(+)},3^{(+)},4^{(-)},5^{(-)}$	640.56	292.14	$1^{(-)},4^{(-)},5^{(-)}$	569.05	285.58	$1^{(-)},4^{(-)},5^{(-)}$	10.84	10.10	$1^{(+)},2^{(+)},3^{(+)}$	3.91	5.01	$1^{(+)},2^{(+)},3^{(+)}$
bou-s-c	100	10000	100	9019.73	284.72	$2^{(-)},4^{(-)},5^{(-)}$	1456.79	37.26	$1^{(+)},4^{(-)}$	1579.05	56.73	$4^{(-)},5^{(-)}$	541.12	31.19	$1^{(+)},2^{(+)},3^{(+)}$	677.47	46.45	$1^{(+)},3^{(+)}$
	100	10000	1000	2016.79	460.67	$2^{(-)},4^{(-)},5^{(-)}$	1451.67	121.58	$1^{(+)},4^{(-)},5^{(-)}$	1503.40	134.60	$4^{(-)},5^{(-)}$	232.06	31.44	$1^{(+)},2^{(+)},3^{(+)}$	252.87	28.84	$1^{(+)},2^{(+)},3^{(+)}$
	100	10000	5000	786.47	308.14	$2^{(+)},3^{(+)},4^{(-)},5^{(-)}$	1404.09	228.27	$1^{(-)},4^{(-)},5^{(-)}$	1489.08	269.61	$1^{(-)},4^{(-)},5^{(-)}$	124.10	36.70	$1^{(+)},2^{(+)},3^{(+)}$	126.24	29.41	$1^{(+)},2^{(+)},3^{(+)}$
	100	10000	15000	589.52	298.05	$2^{(+)},3^{(+)},4^{(-)},5^{(-)}$	1346.87	309.54	$1^{(-)},4^{(-)},5^{(-)}$	1465.54	358.38	$1^{(-)},4^{(-)},5^{(-)}$	72.22	28.64	$1^{(+)},2^{(+)},3^{(+)}$	84.20	32.61	$1^{(+)},2^{(+)},3^{(+)}$

TABLE 7.7: The mean, standard deviation values and statistical tests of the partial offline error for MOEA_D, NSGA-II, SPEA2, NSGA-II with elitism, and SPEA2 with elitism based on the normal distribution.

	n	σ	τ	MOEA_D (1)			NSGA-II (2)			SPEA2 (3)			NSGA-II(we) (4)			SPEA2(we) (5)		
				mean	st	stat	mean	st	stat	mean	st	stat	mean	st	stat	mean	st	stat
uncor	100	100	100	2850.69	502.97	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	7.43	1.75	$1^{(+)},4^{(-)},5^{(-)}$	5.59	2.25	$1^{(+)}$	4.99	2.14	$1^{(+)},2^{(+)}$	4.36	2.76	$1^{(+)},2^{(+)}$
	100	100	1000	190.16	22.90	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	3.41	2.00	$1^{(+)},4^{(-)},5^{(-)}$	1.47	1.24	$1^{(+)},5^{(-)}$	0.45	0.22	$1^{(+)},2^{(+)}$	0.25	0.16	$1^{(+)},2^{(+)},3^{(+)}$
	100	100	5000	45.93	12.35	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	4.73	3.54	$1^{(+)},4^{(-)},5^{(-)}$	2.46	2.54	$1^{(+)},4^{(-)},5^{(-)}$	0.11	0.13	$1^{(+)},2^{(+)},3^{(+)}$	0.00	0.00	$1^{(+)},2^{(+)},3^{(+)}$
	100	100	15000	26.30	12.86	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	4.19	3.49	$1^{(+)},4^{(-)},5^{(-)}$	1.80	2.39	$1^{(+)},4^{(-)},5^{(-)}$	0.06	0.11	$1^{(+)},2^{(+)},3^{(+)}$	0.00	0.00	$1^{(+)},2^{(+)},3^{(+)}$
unc-s-w	100	100	100	1837.47	854.78	$2^{(+)},3^{(+)}$	7839.38	4620.58	$1^{(-)},4^{(-)},5^{(-)}$	7837.07	4647.17	$1^{(-)},4^{(-)},5^{(-)}$	3845.55	2425.39	$2^{(+)},3^{(+)}$	3556.99	2159.38	$2^{(+)},3^{(+)}$
	100	100	1000	1418.01	379.70	$2^{(+)},3^{(+)},4^{(+)},5^{(+)}$	8892.21	1230.48	$1^{(-)},4^{(-)},5^{(-)}$	8980.88	1256.83	$1^{(-)},4^{(-)},5^{(-)}$	4243.08	988.88	$1^{(-)},2^{(+)},3^{(+)}$	4069.83	958.42	$1^{(-)},2^{(+)},3^{(+)}$
	100	100	5000	1258.84	413.82	$2^{(+)},3^{(+)}$	4499.78	1488.58	$1^{(-)},4^{(-)},5^{(-)}$	4521.01	1498.07	$1^{(-)},4^{(-)},5^{(-)}$	2121.57	1216.61	$2^{(+)},3^{(+)}$	2135.18	1218.82	$2^{(+)},3^{(+)}$
	100	100	15000	1083.84	649.82	$2^{(+)},3^{(+)}$	4698.90	1168.47	$1^{(-)},4^{(-)},5^{(-)}$	4705.49	1163.00	$1^{(-)},4^{(-)},5^{(-)}$	1888.21	1330.09	$2^{(+)},3^{(+)}$	2014.72	1485.71	$2^{(+)},3^{(+)}$
bou-s-c	100	100	100	1393.46	183.93	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	18.70	2.02	$1^{(+)},4^{(-)},5^{(-)}$	14.18	1.65	$1^{(+)},4^{(-)},5^{(-)}$	8.20	1.87	$1^{(+)},2^{(+)},3^{(+)}$	7.33	1.69	$1^{(+)},2^{(+)},3^{(+)}$
	100	100	1000	138.05	28.21	$3^{(-)},4^{(-)},5^{(-)}$	17.13	0.85	$4^{(-)},5^{(-)}$	12.47	0.98	$1^{(+)},4^{(-)},5^{(-)}$	2.20	0.63	$1^{(+)},2^{(+)},3^{(+)}$	2.10	1.09	$1^{(+)},2^{(+)},3^{(+)}$
	100	100	5000	52.09	14.34	$3^{(-)},4^{(-)},5^{(-)}$	17.15	1.23	$4^{(-)},5^{(-)}$	12.74	2.07	$1^{(+)},4^{(-)},5^{(-)}$	0.67	0.30	$1^{(+)},2^{(+)},3^{(+)}$	0.38	0.29	$1^{(+)},2^{(+)},3^{(+)}$
	100	100	15000	41.86	20.77	$3^{(-)},4^{(-)},5^{(-)}$	17.39	2.04	$4^{(-)},5^{(-)}$	13.19	2.20	$1^{(+)},4^{(-)},5^{(-)}$	0.35	0.23	$1^{(+)},2^{(+)},3^{(+)}$	0.06	0.07	$1^{(+)},2^{(+)},3^{(+)}$
uncor	100	500	100	1350.69	422.15	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	54.77	29.73	$1^{(+)},4^{(-)},5^{(-)}$	40.24	23.67	$1^{(+)},4^{(-)},5^{(-)}$	7.22	3.63	$1^{(+)},2^{(+)},3^{(+)}$	5.70	3.66	$1^{(+)},2^{(+)},3^{(+)}$
	100	500	1000	183.02	88.65	$3^{(-)},4^{(-)},5^{(-)}$	70.71	52.99	$4^{(-)},5^{(-)}$	51.71	39.88	$1^{(+)},4^{(-)},5^{(-)}$	0.70	0.52	$1^{(+)},2^{(+)},3^{(+)}$	0.20	0.12	$1^{(+)},2^{(+)},3^{(+)}$
	100	500	5000	23.90	14.52	$2^{(+)},3^{(+)},4^{(-)},5^{(-)}$	108.64	59.69	$1^{(-)},4^{(-)},5^{(-)}$	81.33	47.77	$1^{(-)},4^{(-)},5^{(-)}$	0.12	0.11	$1^{(+)},2^{(+)},3^{(+)}$	0.01	0.02	$1^{(+)},2^{(+)},3^{(+)}$
	100	500	15000 "	1.59	1.27	$2^{(+)},3^{(+)},4^{(-)},5^{(-)}$	144.02	64.91	$1^{(-)},4^{(-)},5^{(-)}$	113.76	51.80	$1^{(-)},4^{(-)},5^{(-)}$	0.04	0.13	$1^{(+)},2^{(+)},3^{(+)}$	0.00	0.00	$1^{(+)},2^{(+)},3^{(+)}$
unc-s-w	100	500	100	1884.04	513.97	$2^{(-)},3^{(-)},4^{(-)},5^{(-)}$	18.76	6.17	$1^{(+)},3^{(+)},5^{(+)}$	159.74	61.22	$1^{(+)},2^{(-)},4^{(-)}$	17.92	5.98	$1^{(+)},3^{(+)},5^{(+)}$	71.71	27.52	$1^{(+)},2^{(-)},4^{(-)}$
	100	500	1000	165.71	110.24	$2^{(-)},4^{(-)}$	2.62	1.14	$1^{(+)},3^{(+)},5^{(+)}$	428.28	231.68	$2^{(-)},4^{(-)},5^{(-)}$	1.39	0.84	$1^{(+)},3^{(+)},5^{(+)}$	45.54	20.00	$2^{(-)},3^{(+)},4^{(-)}$
	100	500	5000	13.86	10.22	$2^{(-)},3^{(+)},4^{(-)}$	1.70	1.11	$1^{(+)},3^{(+)},4^{(-)},5^{(+)}$	424.22	348.57	$1^{(-)},2^{(-)},4^{(-)},5^{(-)}$	0.06	0.10	$1^{(+)},2^{(+)},3^{(+)},5^{(+)}$	14.61	13.33	$2^{(-)},3^{(+)},4^{(-)}$
	100	500	15000	2.40	2.04	$3^{(+)},4^{(-)}$	2.22	2.19	$3^{(+)},4^{(-)}$	280.57	302.69	$1^{(-)},2^{(-)},4^{(-)},5^{(-)}$	0.02	0.08	$1^{(+)},2^{(+)},3^{(+)},5^{(+)}$	3.17	4.74	$3^{(+)},4^{(-)}$
bou-s-c	100	500	100	907.16	165.35	$3^{(-)},4^{(-)},5^{(-)}$	100.75	5.39	$4^{(-)},5^{(-)}$	80.71	3.23	$1^{(+)},5^{(-)}$	19.01	1.55	$1^{(+)},2^{(+)}$	13.05	2.21	$1^{(+)},2^{(+)},3^{(+)}$
	100	500	1000	129.70	36.54	$3^{(-)},4^{(-)},5^{(-)}$	106.58	10.22	$3^{(-)},4^{(-)},5^{(-)}$	81.95	6.81	$1^{(+)},2^{(+)},4^{(-)},5^{(-)}$	4.32	1.44	$1^{(+)},2^{(+)},3^{(+)}$	1.70	1.32	$1^{(+)},2^{(+)},3^{(+)}$
	100	500	5000	31.63	20.82	$2^{(+)},4^{(-)},5^{(-)}$	113.69	15.02	$1^{(-)},4^{(-)},5^{(-)}$	85.27	9.19	$4^{(-)},5^{(-)}$	0.86	0.39	$1^{(+)},2^{(+)},3^{(+)}$	0.10	0.12	$1^{(+)},2^{(+)},3^{(+)}$
	100	500	15000	7.40	3.70	$2^{(+)},3^{(+)},5^{(-)}$	119.19	14.98	$1^{(-)},4^{(-)},5^{(-)}$	89.74	13.50	$1^{(-)},4^{(-)},5^{(-)}$	0.33	0.28	$2^{(+)},3^{(+)}$	0.01	0.01	$1^{(+)},2^{(+)},3^{(+)}$

Furthermore, our experiments confirmed that in environments with medium and high frequent changes MOEA_D is outperformed by NSGA-II and SPEA2. However, there is no significant difference between the performance of MOEA_D and classic versions of NSGA-II and SPEA2 when there is sufficient time for algorithms to adapt their solutions according to the new change.

7.5 Conclusion

In this chapter we studied the evolutionary algorithms for the KP where the capacity dynamically changes during the optimisation process. In the introduced dynamic setting, the frequency of changes is determined by τ. The magnitude of changes is chosen randomly either under the uniform distribution $\mathcal{U}(-r, r)$ or under the normal distribution $\mathcal{N}(0, \sigma^2)$. We compared the performance of (1+1) EA, two simple multi-objective approaches with different dominance definitions (MOEA, MOEA_D), the classic versions versions of NSGA-II and SPEA2 as advance multi-objective algorithms, and the effect of customised elitism on their performance. Our experiments in the case of uniform weights verified the previous theoretical studies for (1+1) EA and MOEA [Shi+19]. It is shown that the multi-objective approach, which uses a population in the optimisation, outperforms (1+1) EA. Furthermore, we considered the algorithms in the case of general weights for different classes of instances with a variation of frequencies and magnitudes. Our results illustrated that MOEA does not perform well in the general case due to its dominance procedure. However, MOEA_D, which benefits from a population with a smaller sise and non-dominated solutions, beats (1+1) EA in most cases. On the other hand, in the environments with highly frequent changes, (1+1) EA performs better than the multi-objective approaches. In such cases, the population slows down MOEA_D in reacting to the dynamic change. Selecting MOEA_D as the winner baseline algorithm, we compared its performance with NSGA-II and SPEA2 as advanced evolutionary algorithm. Our results showed that although classic versions of NSGA-II and SPEA2 are significantly better than MOEA_D in many cases, their distribution handling techniques prevent the algorithms to keep track of the optimal solutions. Thus MOEA_D outperform them in low frequent changes. To address this weakness, we improved these algorithms by applying an additional elitism which keeps the best found solution in the population.

Chapter 8

Pareto Optimisation for Dynamic Subset Selection

8.1 Introduction

Subset selection form an important class of problems as many optimisation problems can be modelled by them. In the case of monotone submodular objective functions, greedy algorithms are often able to achieve the best possible worst case approximation guarantee (unless $\mathcal{P} = \mathcal{NP}$). Recently, Pareto optimisation approaches have been investigated for a wide range of subset selection problems. It has been shown that these approaches often achieve the same worst case performance ratio, but obtain solutions of higher quality on real-world benchmarks. It has been shown in [Qia+17] that G-SEMO, which is introduced in Section 2.3.7 and known as POMC in the literature, is able to achieve a $\phi = (\alpha_f/2)(1 - \frac{1}{e^{\alpha_f}})$-approximation where α_f measures the closeness of the considered function f to submodularity. The approximation matches the worst-case performance ratio of the generalised greedy algorithm [ZV16].

In this chapter, we study monotone functions with a dynamic constraint where the constraint bound B changes over time. Such constraint changes reflect real-world scenarios where resources vary during the process. We show that greedy algorithms have difficulties in adapting their current solutions after changes have happened. In particular, we show that there are simple dynamic versions of the classical knapsack problem where adding elements in a greedy fashion when the constraint bound increases over time can lead to an arbitrary bad performance ratio. For the case where constraint bounds decrease over time, we introduce a submodular graph covering problem and show that the considered adaptive generalised greedy algorithm may encounter an arbitrarily bad performance ratio on this problem.

Investigating G-SEMO, we theoretically show that this algorithm obtains for each constraint bound $b \in [0, B]$, a $\phi = (\alpha_f/2)(1 - \frac{1}{e^{\alpha_f}})$-approximation efficiently. Furthermore, when relaxing the bound B to $B^* > B$, ϕ-approximations for all values of $b \in [0, B^*]$ are obtained efficiently. As the complement to our theoretical results,

we perform experiments by considering two real-world problems under two different constraint types and thirty dynamic benchmarks for each of them. We create a baseline by performing our best algorithms for one million generations and statistically compare the results based on offline errors. Moreover, we compare the performance of our algorithms with two additional algorithmic approaches, namely EAMC [Bia+20] and NSGA-II [Deb+02]. EAMC is the most recent in the literature that guarantees a polynomial expected time and NSGA-II is a well-known evolutionary multi-objective algorithm used in many applications.

In the first part of our experiments, by benchmarking the generalised greedy algorithm, its adaptive version and G-SEMO on the influence maximization problem in social networks over sequences of dynamic changes, we show that G-SEMO obtains superior results to the generalised greedy and adaptive generalised greedy algorithms, specifically when the cost function depends on the structure of the graph. We consider the experimental results in four intervals to examine whether the algorithms behave differently during the optimisation process. Here, G-SEMO shows an even more prominent advantage as more dynamic changes are carried out because it is able to improve the quality of its solutions and cater for changes that occur in the future.

To compare these algorithms with EAMC and NSGA-II as the second part of the experiments, we study the maximum coverage problem using two graph benchmarks with different sizes and two cost functions. For the "random" cost, we assign a random value to each node of the graph as their cost value. For the other cost function, called "outdegree," the cost is calculated based on the outdegree of each node. Our results show that EAMC which limits its population size to guarantee the polynomial expected time is not capable of dealing with dynamic changes and performs worse than the other algorithms. NSGA-II which is known to be a high performing evolutionary multi-objective algorithm, on the other hand, beats G-SEMO when the constraint costs are random. However, because of its limited population size and also the effect of the crowding-distance factor in the selection procedure, which forces the population to be distributed along the Pareto front, NSGA-II is inferior to G-SEMO for the outdegree cost function.

This chapter is based on an extended version of a conference paper [Roo+19] which is submitted to the Journal of Artificial Intelligence [Roo+20]. The chapter is structured as follows. In the next section, we formulate the dynamic subset selection problems. In section 8.3, we firstly introduce the algorithms that are studied theoretically and then show that the adaptive generalised greedy algorithm may encounter an arbitrary bad performance ratio even when starting with an optimal solution for the initial budget. In contrast, we show that G-SEMO can maintain ϕ-approximation efficiently. Section 8.4 presents our experimental setting, the results, and the empirical analysis. We discuss our experimental investigations for influence maximisation in social networks in Section 8.4.2. Then extend our investigations by considering

NSGA-II and EAMC facing the benchmarks for maximum coverage problem in Section 8.4.4, and finish with some concluding remarks.

8.2 Problem Formulation

In this chapter we consider optimisation problems in which the cost function and objective functions are monotone and quantified according to their closeness to submodularity. Please note that in this chapter, since we are dealing with sets and elements, capital and small letters are chosen as the notation of sets and elements, respectively.

Chapter 2 presented the different definitions for submodularity and we use the following one in this chapter. For a given set $V = \{v_1, \cdots, v_n\}$, a function $f : 2^V \to \mathbb{R}$ is submodular if for any $X \subseteq Y \subseteq V$ and $v \notin Y$

$$f(X \cup v) - f(X) \geq f(Y \cup v) - f(Y). \tag{8.1}$$

In many applications the function to be optimised, f, comes with a cost function c which is subject to a given cost constraint B. Often the cost function c cannot be evaluated precisely. Hence, the function $\hat{c}$ which is ψ-approximation of c is used [ZV16]. Moreover, according to the submodularity of f, the aim is to find a good approximation instead of finding the optimal solution.

Consider the static version of an optimisation problem defined in 2.

Definition 8.1 (The Static Problem)**.** *Given a monotone objective function $f : 2^V \to \mathbb{R}^+$, a monotone cost function $c : 2^V \to \mathbb{R}^+$ and a budget B, the goal is to compute a solution X such that*

$$X = \arg\max_{Y \subseteq V} f(Y) \; s.t. \; c(Y) \leq B.$$

As for the static case investigated in [Qia+17], we are interested in a ϕ-approximation where $\phi = (\alpha_f/2)(1 - \frac{1}{e^{\alpha_f}})$ depends on the submodularity ratio.

Zhang and Vorobeychik considered the performance of the generalised greedy algorithm [ZV16], given in Algorithm 18, according to the approximated cost function $\hat{c}$. Starting from the empty set, the algorithm always adds the element with the largest objective to cost ratio that does not violate the given constraint bound B.

Let $K_c = \max\{|X| : c(X) \leq B\}$. The optimal solution $\tilde{X}_B$ in these investigations is defined as $\tilde{X}_B = \arg\max\{f(X) \mid c(X) \leq \alpha_c \frac{B(1+\alpha_c^2(K_c-1)(1-\kappa_c)))}{\psi K_c}\}$ where α_c is the submodularity ratio of c. This formulation gives the value of an optimal solution for a slightly smaller budget constraint. The goal is to obtain a good approximation of $f(\tilde{X}_B)$ in this case.

It has been shown in [ZV16] that the generalised greedy algorithm, which adds the item with the highest marginal contribution to the current solution in each step, achieves a $(1/2)(1-\frac{1}{e})$-approximate solution if f is monotone and submodular. [Qia+17] extended these results to objective functions with α_f submodularity ratio and proved that the generalised greedy algorithm obtains a $\phi = (\alpha_f/2)(1-\frac{1}{e^{\alpha_f}})$-approximation. For the remainder of this chapter, we assume $\phi = (\alpha_f/2)(1-\frac{1}{e^{\alpha_f}})$ and are interested in obtaining solutions that are ϕ-approximation for the considered problems.

In this chapter, we study the dynamic version of problem given in Definition 8.1.

Definition 8.2 (The Dynamic Problem)**.** *Let X be a ϕ-approximation for the problem in Definition 8.1. The dynamic problem is given by a sequence of changes where in each change the current budget B changes to $B^* = B + d$, $d \in \mathbb{R}_{\geq -B}$. The goal is to compute a ϕ-approximation X' for each newly given budget B^*.*

The Dynamic Problem evolves over time by the changing budget constraint bounds. Note that every fixed constraint bound gives a static problem and any good approximation algorithm can be run from scratch for the newly given budget. However, the main focus of this chapter are algorithms that can adapt to changes of the constraint bound.

8.3 Theoretical Analysis

G-SEMO and GGA are proven to find a ϕ-approximated solution on the static version of submodular subset selection problems. In this section, we analyse their performance in a dynamic environment. We first consider an extended version of GGA, which is an adaptive version to perform in the dynamic environment. Then we prove the power of G-SEMO in computing ϕ-approximation in static and dynamic versions.

8.3.1 Algorithms

We consider dynamic problems according to Definition 8.2 with $\phi = (\alpha_f/2)(1-\frac{1}{e^{\alpha_f}})$ and are interested in algorithms that adapt their solutions to the new constraint bound B^* and obtain a ϕ-approximation for the new bound B^*. As the generalised greedy algorithm can be applied to any bound B, the first approach would be to run it for the newly given bound B^*. However, this might lead to totally different solutions and adaptation of already obtained solutions might be more beneficial. Furthermore, adaptive approaches that change the solution based on the constraint changes are of interest as they might be faster in obtaining such solutions and/or be able to learn good solutions for the different constraint bounds that occur over time.

Algorithm 18: Generalised Greedy Algorithm

input: Initial budget constraint B.

$X \leftarrow \emptyset$;
$V' \leftarrow V$;
repeat
  $v^* \leftarrow \arg\max_{v \in V'} \frac{f(X \cup v) - f(X)}{\hat{c}(X \cup v) - \hat{c}(X)}$;
  if $\hat{c}(X \cup v^*) \leq B$ **then**
    $X \leftarrow X \cup v^*$;
  $V' \leftarrow V' \setminus \{v^*\}$;
until $V' \leftarrow \emptyset$;
$v^* \leftarrow \arg\max_{v \in V; \hat{c}(v) \leq B} f(v)$;
return $\arg\max_{S \in \{X, v^*\}} f(S)$;

Algorithm 19: Adaptive Generalised Greedy Algorithm

Let B^* be the new budget;
if $B^* < B$ **then**
  while $\hat{c}(X) > B^*$ **do**
    $v^* \leftarrow \arg\min_{v \in X} \frac{f(X) - f(X \setminus \{v\})}{\hat{c}(X) - \hat{c}(X \setminus \{v\})}$;
    $X \leftarrow X \setminus \{v^*\}$;
else if $B^* > B$ **then**
  $V' \leftarrow V \setminus X$;
  repeat
    $v^* \leftarrow \arg\max_{v \in V'} \frac{f(X \cup v) - f(X)}{\hat{c}(X \cup v) - \hat{c}(X)}$;
    if $\hat{c}(X \cup v^*) \leq B^*$ **then**
      $X \leftarrow X \cup v^*$;
    $V' \leftarrow V' \setminus \{v^*\}$;
  until $V' \leftarrow \emptyset$;
$v^* \leftarrow \arg\max_{v \in V; \hat{c}(v) \leq B^*} f(v)$;
return $\arg\max_{S \in \{X, v^*\}} f(S)$;

Based on the generalised greedy algorithm, we introduce the adaptive generalised greedy algorithm. This algorithm is modified from Algorithm 18 in a way that enables it to deal with a dynamic change. Let X be the current solution of the algorithm. When a dynamic change decreases the budget constraint, the algorithm removes items from X according to their marginal contribution, until it achieves a feasible solution. When there is a dynamic increase, this algorithm behaves similarly to the generalised greedy algorithm.

Furthermore, we consider the Pareto optimisation approach G-SEMO (Algorithm 4) presented in Section 2.3.7. G-SEMO is a multi-objective optimisation approach which is proven to perform better than the generalised greedy algorithm in case of local optima [Qia+17]. We reformulate the problem as a bi-objective problem in order to use

G-SEMO as follows:

$$\arg\max_{X\in\{0,1\}^n}(f_1(X), f_2(X)),$$

where:

$$f_1(X) = \begin{cases} -\infty, & \hat{c}(X) > B+1 \\ f(X), & \text{otherwise} \end{cases}, f_2(X) = -\hat{c}(X).$$

This algorithm optimises the cost function and the objective function simultaneously. To achieve this, it uses the concept of dominance to compare two solutions. Solution X_1 dominates X_2, denoted by $X_1 \succeq X_2$, if $f_1(X_1) \geq f_1(X_2) \wedge f_2(X_1) \geq f_2(X_2)$. The dominance is strict, $\succ$, when at least one of the inequalities is strict. G-SEMO produces a population of non-dominated solutions and optimises them during the optimisation process. In each iteration, it chooses solution X randomly from the population and flips each bit of the solution with the probability of $1/n$. It adds the mutated solution X' to the population only if there is no solution in the population that dominates X'. All the solutions which are dominated by X' will be deleted from the population afterward.

Note that we only compute the objective vector $(f_1(X), f_2(X))$ when the solution X is created. This implies that the objective vector is not updated after changes to the constraint bound B. As a consequence solutions whose constraint exceeds the value of $B+1$ for a newly given bound are kept in the population. However, newly produced individuals exceeding $B+1$ for the current bound B are not included in the population as they are dominated by the initial search point 0^n. We are using the value $B+1$ instead of B in the definition of f_1 as this gives the algorithm some look ahead for larger constraint bounds. However, every value of at least B would work for our theoretical analyses. The only drawback would be a potentially larger population size which influences the value $P_{\max}$ in our runtime bounds.

8.3.2 Adaptive Generalised Greedy Algorithm

In this section we analyse the performance of the adaptive generalised greedy algorithm. This algorithm is a modified version of the generalised greedy using the same principle in adding and deleting items. However, in this section we prove that the adaptive generalised greedy algorithm is not able to deal with the dynamic change, i.e., the approximation obtained can become arbitrarily bad during a sequence of dynamic changes.

In order to show that the adaptive generalised greedy algorithm can not deal with dynamic increases of the constraint bound, we consider a special instance of the classical knapsack problem. Note that the knapsack problem is special submodular problem where both the objective and the cost function are linear.

Given $n+1$ items $e_i = (c_i, f_i)$ with cost c_i and value f_i independent of the choice of the other items, we assume there are items $e_i = (1, \frac{1}{n})$, $1 \leq i \leq n/2$, $e_i = (2,1)$, $n/2+1 \leq i \leq n$, and a special item $e_{n+1} = (1,3)$. We have $f_{inc}(X) = \sum_{e_i \in X} f_i$ and $c_{inc}(X) = \sum_{e_i \in X} c_i$ as the linear objective and constraint function, respectively.

Theorem 8.3. *Given the dynamic knapsack problem (f_{inc}, c_{inc}), starting with $B = 1$ and increasing the bound $n/2$ times by 1, the adaptive generalised greedy algorithm computes a solution that has approximation ratio $O(1/n)$.*

Proof. For the constraint $B = 1$ the optimal solution is $\{e_{n+1}\}$. Now let there be $n/2$ dynamic changes where each of them increases B by 1. In each change, the algorithm can only pick an item from $\{e_1, \cdots, e_{n/2}\}$, otherwise it violates the budget constraint. After $n/2$ changes, the budget constraint is $1 + n/2$ and the result of the algorithm is $S = \{e_{n+1}, e_1, \cdots, e_n/2\}$ with $f(S) = 3 + (n/2) \cdot (1/n) = 7/2$ and $c(S) = 1 + n/2$. However, an optimal solution for budget $1 + n/2$ is $S^* = \{e_{n+1}, e_{n/2+1}, \ldots, e_{\frac{3n}{4}}\}$ with $f(S^*) = 3 + \frac{n}{4}$. Hence, the approximation ratio in this example is $(7/2)/(3 + n/4) = O(1/n)$ □

Now we consider the case where the constraint bound decreases over time and show that the adaptive generalised greedy algorithm may also encounter situations where the approximation ratio becomes arbitrarily bad over time.

We consider the following *Graph Coverage Problem*. Let $G = (U, V, E)$ be a bipartite graph with bipartition U and V of vertices with $|U| = n$ and $|V| = m$. The goal is to select a subset $S \subseteq U$ with $|S| \leq B$ such that the number of neighbors of S in V is maximised. Note that the objective function $f(S)$ measuring the number of neighbors of S in V is monotone and submodular.

We consider the graph $G = (U, V, E)$ which consists of k disjoint subgraphs

$$G_i = (U_i = \{u_1^i, \cdots, u_l^i\}, V_i = \{v_1^i, \cdots, v_{2l-2}^i\}, E_i)$$

(see Figure 8.1). Node u_1^i is connected to nodes v_{2j-1}^i, $1 \leq j \leq l-1$. Moreover, each vertex u_j^i, $2 \leq j \leq l$ is connected to two vertices v_{2j-3}^i and v_{2j-2}^i. We assume that $k = \sqrt{n}$ and $l = n/k = \sqrt{n}$.

Theorem 8.4. *Starting with the optimal set $S = U$ and budget $B = n$, there is a specific sequence of dynamic budget reductions such that the solution obtained by the adaptive generalised greedy algorithm has the approximation ratio $O(1/\sqrt{n})$.*

Proof. Let the adaptive generalised greedy algorithm be initialised with $X = U$ and $B = n = kl$. We assume that the budget decreases from n to k where each single decrease reduces the budget by 1. In the first k steps, to change the cost of solution from n to $n-k$, the algorithm removes the nodes u_1^i, $1 \leq i \leq k$, as they have a marginal contribution of 0. Following these steps, all the remaining nodes have the

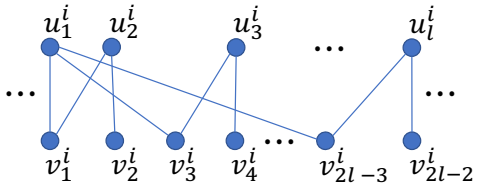


FIGURE 8.1: Single subgraph G_i of $G = (U, V, E)$

same marginal contribution of 2. The solution X of size k obtained by the removal steps of the adaptive generalised greedy algorithm contains k vertices which are connected to $2k$ nodes of V, thus $f(X) = 2k = 2\sqrt{n}$. Such a solution is returned by the algorithm for $B = k$ as the most valuable single node has at most $(l-1) = (\sqrt{n}-1)$ neighbors in V. For $B = k$, the optimal solution $X^* = \{u_1^i \mid 1 \leq i \leq k\}$ has $f(X^*) = k(l-1) = n - \sqrt{n}$. Therefore, the approximation ratio achieved by the adaptive generalised greedy algorithm is upper bounded by $(2\sqrt{n})/(n-\sqrt{n}) = O(1/\sqrt{n})$. □

8.3.3 Pareto Optimisation

In this section we analyse the behaviour of G-SEMO facing a dynamic change. According to Lemma 3 in [Qia+17], for any $X \subseteq V$ and $v^* = \arg\max_{v \notin X} \frac{f(X \cup v) - f(X)}{\hat{c}(X \cup v) - \hat{c}(X)}$, we have

$$f(X \cup v^*) - f(X) \geq \alpha_f \frac{\hat{c}(X \cup v^*) - \hat{c}(X)}{B} \cdot (f(\tilde{X}) - f(X)).$$

We denote by $\delta_{\hat{c}} = \min\{\hat{c}(X \cup v) - \hat{c}(X) \mid X \subseteq V, v \notin X\}$ the smallest contribution of an element to the cost of a solution for the given problem. Moreover, let $P_{\max}$ be the maximum size of G-SEMO's population during the optimisation process.

The following theorem considers the static case and shows that G-SEMO computes a ϕ-approximation efficiently for every budget $b \in [0, B]$.

Theorem 8.5. *Starting from $\{0\}^n$, G-SEMO computes, for any budget $b \in [0, B]$, a $\phi = (\alpha_f/2)(1 - 1/e^{\alpha_f})$-approximate solution after $T = cnP_{\max} \cdot \frac{B}{\delta_{\hat{c}}}$ iterations with the constant probability, where $c \geq 8e + 1$ is a sufficiently large arbitrary constant.*

Proof. For a budget $b \in [0, B]$ and some k, we first consider the number of iterations to find a $(\alpha_f/2)\left(1 - (1 - \frac{\alpha_f}{k})^k\right)$-approximate solution. We consider the largest value of i for which there is a solution X in the population where $\hat{c}(X) \leq i < b$ and

$$f(X) \geq \left(1 - \left(1 - \alpha_f \frac{i}{bk}\right)^k\right) \cdot f(\tilde{X}_b)$$

holds for some k. Initially, it is true for $i = 0$ with $X = \{0\}^n$. We now show that adding v^* to the current solution has the desired contribution to achieve a ϕ-approximate solution. Let $X \subseteq V$ and $v^* = \arg\max_{v \notin X} \frac{f(X \cup v) - f(X)}{\hat{c}(X \cup v) - \hat{c}(X)}$.

Assume that

$$f(X) \geq \left(1 - \left(1 - \alpha_f \frac{i}{bk}\right)^k\right) \cdot f(\tilde{X}_b)$$

holds for some $\hat{c}(X) \leq i < b$ and k. Then adding v^* leads to

$$f(X \cup v^*) \geq \left(1 - \left(1 - \alpha_f \frac{i + \hat{c}(X \cup v^*) - \hat{c}(X)}{b(k+1)}\right)^{k+1}\right) \cdot f(\tilde{X}_b).$$

This process only depends on the quality of X and is independent of its structure. Starting from $\{0\}^n$, if the algorithm carries out such steps at least $b/\delta_{\hat{c}}$ times, it reaches a solution X such that

$$\begin{aligned} f(X \cup v^*) &\geq \left(1 - \left(1 - \alpha_f \frac{b}{bk^*}\right)^{k^*}\right) \cdot f(\tilde{X}_b) \\ &\geq \left(1 - \frac{1}{e^{\alpha_f}}\right) \cdot f(\tilde{X}_b). \end{aligned}$$

Considering item $z = \arg\max_{v \in V : \hat{c}(v) \leq b} f(v)$, by submodularity and $\alpha \in [0,1]$ we have $f(X \cup v^*) \leq (f(X) + f(z))/\alpha_f$.

This implies that

$$\max\{f(X), f(z)\} \geq (\alpha_f/2) \cdot (1 - \frac{1}{e^{\alpha_f}}) \cdot f(\tilde{X}_b).$$

We consider $T = cnP_{\max}B/\delta_{\hat{c}}$ iterations of the algorithm and analyse the success probability within T steps. To have a successful mutation step where v^* is added to the currently best approximate solution, the algorithm has to choose the right individual in the population, which happens with probability at least $1/P_{\max}$. Furthermore, the single bit corresponding to v^* has to be flipped which has the probability at least $1/(en)$. We call such a step a success. Let random variable $Y_j = 1$ when there is a success in iteration j of the algorithm and $Y_j = 0$, otherwise. Thus, we have

$$\Pr(Y_j = 1) \geq \frac{1}{en} \cdot \frac{1}{P_{\max}}$$

as long as a ϕ-approximation for bound b has not been obtained.

Furthermore, let Y_i^*, $1 \leq i \leq T$, be mutually independent random binary variables with $\Pr[Y_i^* = 1] = \frac{1}{enP_{\max}}$ and $\Pr[Y_i^* = 0] = 1 - \frac{1}{enP_{\max}}$. For the expected value of the random variable $Y^* = \sum_{j=1}^{T} Y_j^*$ we have:

$$E[Y^*] = \frac{T}{enP_{\max}} = \frac{cB}{e\delta_{\hat{c}}} \geq \frac{cb}{e\delta_{\hat{c}}}.$$

We use Lemma 1 in [DHK11] for moderately correlated variables which allows the use of the following Chernoff bound

$$\begin{aligned}\Pr\left(Y < (1-\delta)E[Y^*]\right) &\leq \Pr\left(Y^* < (1-\delta)E[Y^*]\right)\\ &\leq e^{-E[Y^*]\delta^2/2}.\end{aligned} \tag{8.2}$$

Using Equation 8.2 with $\delta = (1-\frac{e}{c})$, we bound the probability of not finding a ϕ-approximation of $\tilde{X}_b$ in time $T = cnP_{\max}B/\delta_{\hat{c}}$ by

$$\begin{aligned}\Pr(Y \leq \frac{b}{\delta_{\hat{c}}}) &\leq e^{-\frac{(c-e)^2B}{2ce\delta_{\hat{c}}}} \leq e^{-\frac{(c/2)^2B}{2ce\delta_{\hat{c}}}}\\ &\leq e^{-\frac{cB}{8e\delta_{\hat{c}}}} \leq e^{-\frac{B}{\delta_{\hat{c}}}}.\end{aligned}$$

Using the union bound and taking into account that there are at most $B/\delta_{\hat{c}}$ different values for b to consider, the probability that there is a $b \in [0,B]$ for which no ϕ-approximation has been obtained is upper bounded by $\frac{B}{\delta_{\hat{c}}} \cdot e^{-\frac{B}{\delta_{\hat{c}}}}$.

This implies that G-SEMO finds a $(\alpha_f/2)(1-\frac{1}{e^{\alpha_f}})$-approximate solution with probability at least $1-\frac{B}{\delta_{\hat{c}}} \cdot e^{-\frac{B}{\delta_{\hat{c}}}}$ for each $b \in [0,B]$. □

Note that in case of $B/\delta_{\hat{c}} \geq \log n$, the probability of achieving a ϕ-approximation for every $b \in [0,B]$ is $1-o(1)$. In order to achieve a probability of $1-o(1)$ for any possible change, we can run the algorithm for $T' = cnP_{\max} \cdot \max\{\log n, \frac{B}{\delta_{\hat{c}}}\}$, $c \geq 8e+1$, iterations.

Now we consider the performance of G-SEMO in the dynamic version of the problem. In this version, it is assumed that G-SEMO has achieved a population which includes a ϕ-approximation for all budgets $b \in [0,B]$. Reducing the budget from B to B^* implies that a ϕ-approximation for the newly given budget B^* is already contained in the population.

Consideration must be given to the case where the budget increases. Assume that the budget changes from B to $B^* = B+d$ where $d > 0$. We analyse the time until G-SEMO has updated its population such that it contains for any $b \in [0,B^*]$ a ϕ-approximate solution.

We define

$$\begin{aligned}I_{\max}(b,b') = \max\{i \in [0,b] \mid \exists X \in P, \hat{c}(X) \leq i\\ \wedge f(X) \geq \left(1-\left(1-\alpha_f\frac{i}{bk}\right)^k\right) \cdot f(\tilde{X}_b)\\ \wedge f(X) \geq \left(1-\left(1-\alpha_f\frac{i}{b'k'}\right)^{k'}\right) \cdot f(\tilde{X}_{b'})\}\end{aligned}$$

for some k and k'. The notion of $I_{\max}(b, b')$ enables us to correlate progress in terms of obtaining a ϕ-approximation for budgets b and b'.

Theorem 8.6. *Let G-SEMO have a population P such that, for every budget $b \in [0, B]$, there is a ϕ-approximation in P. After changing the budget to $B^* > B$, G-SEMO has computed a ϕ-approximation with probability $\Omega(1)$ within $T = cnP_{\max}\frac{d}{\delta_{\hat{c}}}$ steps for every $b \in [0, B^*]$.*

Proof. Let P denote the current population of G-SEMO in which, for any budget $b \leq B$, there is a $\left(1 - (1 - \frac{\alpha_f}{k})^k\right)$-approximate solution for some k.

Let X be the solution corresponding to $I_{\max}(B, B^*)$. Let $v^* = \arg\max_{v \notin X} \frac{f(X \cup v) - f(X)}{\hat{c}(X \cup v) - \hat{c}(X)}$ be the item with the highest marginal contribution which could be added to X and $X' = X \cup v^*$. According to Lemma 3 and Theorem 2 in [Qia+17] and the definition of $I_{\max}(B, B^*)$, we have

$$f(X') \geq \left(1 - (1 - \alpha_f \frac{I_{\max} + \hat{c}(X') - \hat{c}(X)}{Bk})^k\right) \cdot f(\tilde{X}_B)$$

and

$$f(X') \geq \left(1 - (1 - \alpha_f \frac{I_{\max} + \hat{c}(X') - \hat{c}(X)}{B^* k'})^{k'}\right) \cdot f(\tilde{X}_{B^*}).$$

This implies that adding v^* to X violates the budget constraint B, otherwise we would have a greater value for $I_{\max}$.

If $I_{\max} + \hat{c}(X') - \hat{c}(X) \geq B^*$, then, similar to the proof of Theorem 8.5, we have

$$\max\{f(X), f(z)\} \geq (\alpha_f/2) \cdot \left(1 - \frac{1}{e^{\alpha_f}}\right) \cdot f(\tilde{X}_{B^*}).$$

Otherwise, we have

$$f(X') \geq \left(1 - (1 - \alpha_f \frac{B}{B^* k'})^{k'}\right) \cdot f(\tilde{X}_{B^*}).$$

From this point, the argument in the proof of Theorem 8.5 holds, i.e., G-SEMO obtains, for each value $b \in [B, B^*]$, a ϕ-approximation after $\frac{d}{\delta_{\hat{c}}}$ successes.

Hence, after $T = cnP_{\max} d/\delta_{\hat{c}}$ iterations, for all $b \in [B, B^*]$ with probability $1 - \frac{d}{\delta_{\hat{c}}} \cdot e^{-\frac{d}{\delta_{\hat{c}}}}$, we have a $\phi = (\alpha_f/2)(1 - \frac{1}{e^{\alpha_f}})$-approximation in the population. □

Note that if the dynamic change is sufficiently large such that $\frac{d}{\delta_{\hat{c}}} \geq \log n$, then the probability of having obtained, for every budget $b \in [0, B^*]$, a ϕ-approximation increases to $1 - o(1)$. A success probability of $1 - o(1)$ can be obtained for this magnitude of changes by giving the algorithm time $T' = cnP_{\max} \max\{\log n, \frac{d}{\delta_{\hat{c}}}\}$, $c \geq 8e + 1$.

A special class of known submodular problems is the maximisation of a function with a cardinality constraint. In this case, the constraint value can take on at most $n+1$ different values and we have $P_{\max} \leq n+1$. Furthermore, we have $\delta = 1$ which leads to the following two corollaries.

Corollary 8.7. *Consider the static problem with cardinality constraint bound B. G-SEMO computes, for every budget $b \in [0, B]$, a ϕ-approximation within $T = cn^2 \cdot \max\{B, \log n\}$, $c \geq 8e+1$, iterations with probability $1 - o(1)$.*

Corollary 8.8. *Consider the dynamic problem with a cardinality constraint B. Assume that P contains a ϕ-approximation for every $b \in [0, B]$. Then after increasing the budget to B^*, G-SEMO computes, for every budget $b \in [0, B^*]$, a ϕ-approximation in time $T = cn^2 \max\{d, \log n\}$, $c \geq 8e+1$ and $d = |B^* - B|$, with probability $1 - o(1)$.*

8.4 Experimental Investigations

In this section, we experimentally compare the performance of our algorithms on dynamic variants of problems, where the constraint bound changes over time. We first analyse the practical performance of three algorithms that have been investigated theoretically in the previous section, the generalised greedy algorithm (GGA), the adaptive generalised greedy algorithm (AdGGA) and G-SEMO, on the dynamic submodular influence maximisation problem [Qia+17; ZV16]. In addition to the plain G-SEMO described in Algorithm 4, we also consider another version in which the algorithm has a warm-up phase. Starting from a zero solution, G-SEMO$^{\text{wp}}$ performs ten thousand generations before the first change happens, which gives it more time to build a population that is prepared for the following changes.

Afterward, we include two more algorithms. EAMC, introduced in [Bia+20], is a newer algorithm that theoretically guarantees ϕ-approximation in polynomial expected time $2en^2(n+1)$. The other algorithm is a version of NSGA-II that benefits from a specific elitism to keep track of the best-found solution in a dynamic environment. We compare these two algorithms with G-SEMO$^{\text{wp}}$, GGA and AdGGA on the dynamic submodular maximum coverage problem. The following section describes our experimental setting.

8.4.1 Experimental Setting

We build our dynamic benchmark based on the approach in Chapter 7. We have two problems, and for each problem, we consider two different cost functions. Thus, we study four types of instances. Each instance has an initial, a maximum and a minimum budget denoted by B_{init}, $B_{\max}$ and $B_{\min}$, respectively. Every τ evaluation, a dynamic change adds the value of $\delta \in [-r, r]$, which is chosen uniformly at random, to the current budget $B \in [B_{\max}, B_{\min}]$. In other words, the iterative algorithms have τ evaluations to find a solution for budget constraint B, before the next change happens. We consider two hundred changes for each run, and there are thirty different

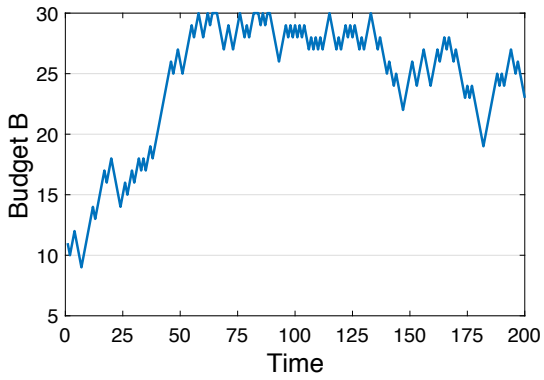


FIGURE 8.2: Budget over time for dynamic problems

sequences of random changes produced for each instance. An example of how the budget values change over time with $B_{\text{init}} = 10$ and $\delta \in \{-1, 1\}$ is shown in Figure 8.2. For the baseline in the influence maximisation and the maximum coverage problems, we run G-SEMO$^{\text{wp}}$ and NSGA-II for all occurred budget constraints for one million generations, respectively.

Algorithms start with the initial budget constraint B_{init}. To calculate the error for each algorithm, let s^i denote the best-found solution right before the change i happens, and s^i_{b} be the solution found by the baseline algorithm. For each dynamic change, we record error e_i as : $e_i = f(s^i_{\text{b}}) - f(s^i)$. Then for each interval that contains m changes, the partial offline error is $\sum_{i=1}^{m} e_i/m$.

We compare the performance of algorithms for each instance according to the thirty partial offline error values. To establish a statistical comparison of the results among different algorithms, we use a multiple comparisons test. In particularity, we focus on the method that compares a set of algorithms. For statistical validation, we use the Kruskal-Wallis test with 95% confidence. Afterwards, we apply the Bonferroni post-hoc statistical procedures that are used for multiple comparisons of a control algorithm against two or more other algorithms. For more detailed descriptions of the statistical tests, we refer the reader to [CF09].

Our results are summarised in the Tables 8.1 and 8.2. The columns represent the algorithms with the corresponding mean value and standard deviation. Note, $X^{(+)}$ is equivalent to the statement that the algorithm in the column outperformed algorithm X, and $X^{(-)}$ is equal to the statement that X outperformed the algorithm in the given column. If the algorithm X does not appear, this means that no significant difference was observed between the algorithms.

8.4.2 The Influence Maximisation Problem

The influence maximisation problem aims to identify a set of most influential users in a social network. Given a directed graph $G = (V, E)$ where each node represents a user. Each edge $(u, v) \in E$ has assigned an edge probability $p_{u,v}((u, v) \in E)$. The

probability $p_{u,v}$ corresponds to the strengths of influence from user u to user v. The goal is to find a subset $X \subseteq V$ such that the expected number of activated nodes from X, $IC(X)$, is maximised. Given a cost function c and a budget B the submodular optimisation problem is formulated as $\arg\max_{X \subseteq V} E[|IC(X)|]$ s.t. $c(X) \leq B$. To calculate $E[|IC(X)|]$ in our experiments, we simulate the random process of influence of solution X for 500 times independently, and use the average as the value of $E[|IC(X)|]$.

We consider two types of cost functions. The routing constraint takes into account the costs of visiting nodes whereas the cardinality constraint counts the number of chosen nodes. For both cost functions, the constraint is met if the cost is at most B. For more detailed descriptions of the influence maximisation through a social network problem, we refer the reader to [KKT15; Qia+17; ZV16].

To build the dynamic benchmark, as described in Section 8.4.1, we assume that the initial constraint bound is $B_{\text{init}} = 10$, which stays within the interval $[5, 30]$. $\delta \in \{-1, 1\}$ is chosen uniformly at random and we consider four values for $\tau \in \{100, 1000, 5000, 10000\}$. In G-SEMO$^{\text{wp}}$, we consider the option of G-SEMO having a warm-up phase where there are no dynamic changes for the first 10000 evaluations. This allows G-SEMO$^{\text{wp}}$ to optimise for an extended period for the initial bound. It should be noted that the number of evaluations in the warm-up phase and our choices of τ are relatively small compared to the choice of $10eBn^2$ used in [Qia+17] for optimising the static problem with a given fixed bound B. The results are shown in Table 8.1. For this problem, we divide the experiment, which contains a sequence of two hundred changes, into 4 parts and we perform the statistical tests for each part separately. In this way, we show how the iterative algorithms perform during the optimisation process.

Empirical Analysis

We first investigate the influence maximisation for the routing constraint that is based on the simulated networks as done for the static case in [Qia+17]. We consider a social network with 400 nodes that are built using the popular Barabasi-Albert (BA) model [AB02] with edge probability $p = 0.1$. The routing network is based on the Erdos-Renyi (ER) model [ER59] where each edge is presented with probability $p = 0.02$. Nodes are placed randomly in the plane and the edge costs are given by Euclidean distances. Furthermore, each chosen node has a cost of 0.1.

As it can be observed in Table 8.1, in the instances with $\tau = 100$, both greedy algorithms are significantly better than POMCs. When the frequency of changes is that high, G-SEMO$^{\text{wp}}$ is not even able to keep reducing the mean of errors in the third and fourth intervals. Furthermore, the impact of the warm-up phase is only noticeable in the first interval and fades when the plain G-SEMO has more time to adapt its population.

TABLE 8.1: The mean, standard deviation values and statistical tests of the partial offline error for GGA, AdGGA, G-SEMO, and G-SEMO$^{\text{wp}}$ for the dynamic influence maximisation problem.

	r	τ	interval	GGA (1) mean	st	stat	AdGGA (2) mean	st	stat	G-SEMO (3) mean	st	stat	G-SEMO$^{\text{wp}}$ (4) mean	st	stat
Rout	1	100	1-50	10.05	1.57	$3^{(+)},4^{(+)}$	11.53	3.76	$3^{(+)},4^{(+)}$	57.93	20.13	$1^{(-)},2^{(-)},4^{(-)}$	29.25	9.98	$1^{(-)},2^{(-)},3^{(+)}$
constraint	1	100	51-100	9.98	1.69	$3^{(+)},4^{(+)}$	12.36	4.91	$3^{(+)},4^{(+)}$	40.88	16.20	$1^{(-)},2^{(-)}$	27.65	9.89	$1^{(-)},2^{(-)}$
	1	100	100-151	9.73	1.48	$3^{(+)},4^{(+)}$	12.25	4.04	$3^{(+)},4^{(+)}$	36.96	11.91	$1^{(-)},2^{(-)}$	29.11	11.18	$1^{(-)},2^{(-)}$
	1	100	151-200	9.72	1.57	$3^{(+)},4^{(+)}$	14.22	5.39	$3^{(+)},4^{(+)}$	35.96	13.74	$1^{(-)},2^{(-)}$	29.75	12.41	$1^{(-)},2^{(-)}$
	1	1000	1-50	10.05	1.57	$3^{(+)},4^{(+)}$	11.53	3.76	$3^{(+)},4^{(+)}$	24.73	5.17	$1^{(-)},2^{(-)}$	18.80	4.63	$1^{(-)},2^{(-)}$
	1	1000	51-100	9.98	1.69	$3^{(+)},4^{(+)}$	12.36	4.91		14.10	3.96	$1^{(-)}$	12.95	4.21	$1^{(-)}$
	1	1000	100-151	9.73	1.48	$2^{(+)},3^{(+)}$	12.25	4.04	$1^{(-)}$	12.38	3.57	$1^{(-)}$	11.68	4.27	
	1	1000	151-200	9.72	1.57	$2^{(+)}$	14.22	5.39	$1^{(-)},3^{(-)},4^{(-)}$	11.19	5.54	$2^{(+)}$	10.55	4.67	$2^{(+)}$
	1	5000	1-50	10.05	1.57		11.53	3.76	$4^{(-)}$	10.74	2.33		9.16	2.49	$2^{(+)}$
	1	5000	51-100	9.98	1.69	$3^{(-)},4^{(-)}$	12.36	4.91	$3^{(-)},4^{(-)}$	4.32	1.94	$1^{(+)},2^{(+)}$	3.96	2.18	$1^{(+)},2^{(+)}$
	1	5000	100-151	9.73	1.48	$3^{(-)},4^{(-)}$	12.25	4.04	$3^{(-)},4^{(-)}$	3.45	1.98	$1^{(+)},2^{(+)}$	3.38	2.28	$1^{(+)},2^{(+)}$
	1	5000	151-200	9.72	1.57	$3^{(-)},4^{(-)}$	14.22	5.39	$3^{(-)},4^{(-)}$	2.64	2.42	$1^{(+)},2^{(+)}$	2.65	2.79	$1^{(+)},2^{(+)}$
	1	10000	1-50	10.05	1.57	$3^{(-)},4^{(-)}$	11.53	3.76	$3^{(-)},4^{(-)}$	6.84	1.91	$1^{(+)},2^{(+)}$	5.78	2.20	$1^{(+)},2^{(+)}$
	1	10000	51-100	9.98	1.69	$3^{(-)},4^{(-)}$	12.36	4.91	$3^{(-)},4^{(-)}$	2.39	2.68	$1^{(+)},2^{(+)}$	1.50	2.05	$1^{(+)},2^{(+)}$
	1	10000	100-151	9.73	1.48	$3^{(-)},4^{(-)}$	12.25	4.04	$3^{(-)},4^{(-)}$	1.91	1.91	$1^{(+)},2^{(+)}$	1.21	2.35	$1^{(+)},2^{(+)}$
	1	10000	151-200	9.72	1.57	$3^{(-)},4^{(-)}$	14.22	5.39	$3^{(-)},4^{(-)}$	1.27	2.42	$1^{(+)},2^{(+)}$	1.08	2.42	$1^{(+)},2^{(+)}$
Card	1	100	1-50	1.18	0.47	$3^{(+)},4^{(+)}$	1.25	0.57	$3^{(+)},4^{(+)}$	68.97	12.46	$1^{(-)},2^{(-)},4^{(-)}$	36.68	13.13	$1^{(-)},2^{(-)},3^{(+)}$
constraint	1	100	51-100	1.00	0.55	$3^{(+)},4^{(+)}$	1.15	0.62	$3^{(+)},4^{(+)}$	51.53	12.07	$1^{(-)},2^{(-)},4^{(-)}$	34.84	13.08	$1^{(-)},2^{(-)},3^{(+)}$
	1	100	100-151	0.90	0.57	$3^{(+)},4^{(+)}$	0.97	0.63	$3^{(+)},4^{(+)}$	46.25	11.26	$1^{(-)},2^{(-)}$	33.98	13.38	$1^{(-)},2^{(-)}$
	1	100	151-200	0.75	0.65	$3^{(+)},4^{(+)}$	0.79	0.72	$3^{(+)},4^{(+)}$	42.91	12.59	$1^{(-)},2^{(-)}$	32.31	13.04	$1^{(-)},2^{(-)}$
	1	1000	1-50	1.18	0.47	$3^{(+)},4^{(+)}$	1.25	0.57	$3^{(+)},4^{(+)}$	31.53	7.18	$1^{(-)},2^{(-)}$	22.93	4.58	$1^{(-)},2^{(-)}$
	1	1000	51-100	1.00	0.55	$3^{(+)},4^{(+)}$	1.15	0.62	$3^{(+)},4^{(+)}$	13.97	5.58	$1^{(-)},2^{(-)}$	12.31	4.36	$1^{(-)},2^{(-)}$
	1	1000	100-151	0.90	0.57	$3^{(+)},4^{(+)}$	0.97	0.63	$3^{(+)},4^{(+)}$	10.22	4.59	$1^{(-)},2^{(-)}$	9.55	3.14	$1^{(-)},2^{(-)}$
	1	1000	151-200	0.75	0.65	$3^{(+)},4^{(+)}$	0.79	0.72	$3^{(+)},4^{(+)}$	8.28	5.48	$1^{(-)},2^{(-)}$	7.82	4.29	$1^{(-)},2^{(-)}$
	1	5000	1-50	1.18	0.47	$3^{(+)},4^{(+)}$	1.25	0.57	$3^{(+)},4^{(+)}$	10.25	2.65	$1^{(-)},2^{(-)}$	7.88	2.02	$1^{(-)},2^{(-)}$
	1	5000	51-100	1.00	0.55	$3^{(+)}$	1.15	0.62		1.80	1.02	$1^{(-)}$	1.29	0.95	
	1	5000	100-151	0.90	0.57		0.97	0.63		1.59	1.29		1.29	0.97	
	1	5000	151-200	0.75	0.65		0.79	0.72		1.09	1.31		1.02	1.23	
	1	10000	1-50	1.18	0.47	$3^{(+)},4^{(+)}$	1.25	0.57	$3^{(+)},4^{(+)}$	5.40	1.41	$1^{(-)},2^{(-)}$	4.17	1.12	$1^{(-)},2^{(-)}$
	1	10000	51-100	1.00	0.55	$4^{(-)}$	1.15	0.62	$3^{(-)},4^{(-)}$	0.68	0.63	$2^{(+)}$	0.40	0.58	$1^{(+)},2^{(+)}$
	1	10000	100-151	0.90	0.57	$4^{(-)}$	0.97	0.63	$4^{(-)}$	0.59	0.56		0.31	0.62	$1^{(+)},2^{(+)}$
	1	10000	151-200	0.75	0.65	$3^{(-)},4^{(-)}$	0.79	0.72	$3^{(-)},4^{(-)}$	0.19	0.37	$1^{(+)},2^{(+)}$	0.16	0.60	$1^{(+)},2^{(+)}$

Considering the instances with medium-frequency changes, we see a remarkable improvement in the performance of G-SEMO and G-SEMO$^{\text{wp}}$. Although GGA and AdGGA are still better in the first fifty changes of $\tau = 1000$, the results show that G-SEMO and G-SEMO$^{\text{wp}}$ outperform AdGGA in the final interval and are not significantly worse than GGA anymore. In the frequency of $\tau = 5000$, GGA does not beat POMCs in the first interval either. This shows the exceptional ability of G-SEMO algorithms in providing a population so that efficiently adapts to an environment with dynamic changes in such a short time. Note that in routing constraint, there is no significant difference in the performance of G-SEMO and G-SEMO$^{\text{wp}}$ when the frequency of changes is medium. However, the results depict that in some cases, such as the third interval of $\tau = 1000$ and the first interval of $\tau = 5000$, G-SEMO$^{\text{wp}}$ outperforms AdGGA while G-SEMO is unable to do so.

When the optimisation interval increases to $\tau = 10000$, G-SEMO and G-SEMO$^{\text{wp}}$ outperform GGA and AdGGA in all the intervals. This shows that 10000 evaluations are enough for G-SEMO to perform better than the greedy algorithms.

It should be noticed that the routing cost is affected by the structure of the graph. Let there are some nodes in the graph that can influence a considerable number of other nodes. Because of the routing constraint and their distance, the greedy algorithms might not select them together in a single solution. In other words, the

greedy behaviour of GGA and AdGGA might be against their performance in routing constraint. Such cases do not appear when the constraint is on the number of selected nodes in a solution. Hence, the greedy algorithms should behave much better in cardinality constraint. While our theoretical analyses show the outperformance of G-SEMO, the interesting question is how much does it take for G-SEMO and G-SEMO$^{\text{wp}}$ to track the dynamic changes in practice.

To consider the cardinality constraint, we use the social news data which is collected from the social news aggregator Digg. The Digg dataset contains stories submitted to the platform over a period of a month, and IDs of users who voted on the popular stories. The data consist of two tables that describe friendship links between users and the anonymised user votes on news stories [HL12]. As in [Qia+17], we use the preprocessed data with 3523 nodes and 90244 edges, and the estimated edge probabilities from the user votes based on the method in [BBM12].

While the effect of the warm-up phase is still visible in the first two intervals, similar to the routing constraint, G-SEMO and G-SEMO$^{\text{wp}}$ cannot beat greedy algorithms in cardinality constraint and high-frequency changes. The statistical advantage of greedy algorithms holds even when τ increases to 1000. However, a significant improvement happens in $\tau = 5000$. The results show that greedy algorithms lose their advantage against G-SEMO$^{\text{wp}}$ after the first fifty changes. But G-SEMO needs more time to reach the other algorithms. The fact that evolutionary algorithms do not show a significant performance until $\tau = 5000$ shows the effectiveness of the greedy approaches in situations where the constraint is not affected by the structure of the graph, as discussed above.

Contrary to the routing constraint, G-SEMO and G-SEMO$^{\text{wp}}$ are outperformed by the greedy algorithms in the first interval, even when $\tau = 10000$. However, the warm-up phase helps G-SEMO$^{\text{wp}}$ to take the lead and outperform other algorithms from the second interval. On the other hand, it takes three intervals for the plain G-SEMO to show statistically better results than the greedy algorithms. Note that the warm-up phase in only 10000 evaluations. Thus, having a prepared population, even for 10000 evaluations for such a complicated benchmark, highly improves the results of evolutionary algorithms in dynamic environment.

8.4.3 EAMC and NSGA-II with elitism

In this section, we explain two more algorithms that we study experimentally in addition to the algorithms in previous sections. EAMC (Algorithm 20) has been introduced in [Bia+20] and has been proven to find a ϕ-approximate solution in expected time $2en^2(n+1)$. While the definition of dominance is the only factor to control the population size in G-SEMO, EAMC keeps only two solutions per each possible solution size. Thus, the population size does not increase to more than $2n$ in EAMC.

Algorithm 20: EAMC Algorithm

```
Update B;
Update P;
while stopping criterion not met do
    Select X from P uniformly at random;
    X' ← flip each bit of X with probability 1/n;
    if ĉ(X') ≤ B then
        i ← |X'|;
        if bin(i) ≠ ∅ then
            P ← P ∪ {X'}, U^i ← V^i ← X';
        else
            if g(X') > g(U^i) then
                U^i ← X';
            if f(X') > f(V^i) then
                V^i ← X';
        P ← (P \ bin(i)) ∪ {U^i} ∪ {V^i};
    t = t + 1;
```

In addition to $f(X)$, this algorithm also uses another fitness function as follow:

$$g(X) = \begin{cases} f(X), & |X| = 0 \\ f(X)/(1 - e^{-\alpha_f \hat{c}(X)/B}), & \text{otherwise.} \end{cases}$$

Moreover, function $\text{bin}(i)$ returns solutions in the population with size i. The polynomial bound for EAMC is the results of specific definition of $g(X)$. Note that according to this definition, one needs submodularity ratio of f to use EAMC which computing its exact value might be difficult. However, [Bia+20] showed that using a lower bound α for α_f results in the approximation ratio of $(\alpha_f/2)(1 - 1/e^{\alpha})$. To adopt EAMC to the dynamic environment, we update the population after each dynamic change. Since EAMC keeps and compares solutions according to their size, recording infeasible solutions can cause removing the feasible ones. Thus, the update process only removes solutions that become infeasible after the change.

The other algorithm that we use in this section is the same version of NSGA-II with additional elitism presented in Chapter 7. NSGA-II, Algorithm 5 as explained in Chapter 2, sorts solutions based on non-dominated fronts based on the objective values such that each solutions in front $\mathcal{F}_i$, $i > 1$, is dominated by at least one solution in $\mathcal{F}_{i-1}$. It also computes the distance between solutions in the same front, called crowding distance, and uses this to achieve well-distributed solutions. Finally, front ranks and crowding distance is used to generate the next generation. As we showed in Chapter 7, the plain version of NSGA-II loses the best-found solution during optimisation of the dynamic knapsack problem. That is the result of original approach

which prioritise the distribution of solutions. The additional elitism locates the best-found solution according to the budget constraint in each generation and increases its crowding distance. Hence, it assures that this solution is not removed from the population because of the distribution factor. We use f_N and c_N as the objective functions and set the population size twenty. To prepare this algorithm for the upcoming changes, we let it to keep solutions with costs up to $B + \delta$ and we set high penalty to solutions that violate this bound. Hence, we define f_N and c_N as follow:

$$c_N(X) = \begin{cases} c(X) & \text{if } c(x) \leq B + \delta \\ c(X) + (n \cdot c_{\max} + 1) \cdot h(X) & \text{otherwise} \end{cases} \tag{8.3}$$

and

$$f_N(X) = \begin{cases} f(X) & \text{if } c(x) \leq B + \delta \\ f(X) + (n \cdot f_{\max} + 1) \cdot h(X) & \text{otherwise,} \end{cases} \tag{8.4}$$

where $c_{\max}$, $f_{\max}$, and $h(X)$ are maximum possible cost, maximum possible fitness, and the amount of violation, respectively.

8.4.4 The Maximum Coverage Problem

We use the maximum coverage problem to compare the performance of AdGGA, GGA, G-SEMO$^{\text{wp}}$, EAMC, and NSGA-II, which have been described in previous sections. In this problem, a set of elements U and a collection $V = \{S_1, S_2, \cdots, S_n\}$ of subsets of U are given. Considering a monotone cost function c, the goal is to select subsets from V such that their union cover the maximum number of elements in U, while the cost of this selection do not exceed the budget constraint B. To be more precise, we are looking for $\arg\max_{X \subseteq V} f(X) = |\bigcup_{S_i \in X} S_i|$ such that $c(X) \leq B$. We use directed graphs as the benchmarks for this problem. Each node p represents a subset $S_p \in V$ that contains node p and all of its adjacent nodes. We use two types of cost functions as defined by [Bia+20]. The "outdegree" cost of node p is calculated as $o(p) = 1 - \max\{d(p) - q, 0\}$, where $d(p)$ is the out degree of p and q is a constant set to six. For "random" cost function we assigned a random positive cost value in $(0, 1]$ to each node. Cost of a selected set of nodes X is $c(X) = \sum_{p \in X} o(p)$. Our benchmarks are two graphs originally generated for maximum independent set problem [Xu+07]. frb35-17-mis is a graph with 450 nodes and 17,827 edges, and the graph of frb30-15-mis has 595 nodes and 27,856 edges.

To apply the dynamic changes, similar to Section 8.4.2, we generated thirty files for each constraint that includes two hundred random numbers. However, the magnitudes of changes, budget intervals, and initial budget are chosen according to each cost function. The budget for "outdegree" is initially set to 500 that is bounded by the $B_{\min} = 250$ and $B_{\max} = 750$ during the optimisation process, and δ is a random integer within $[-20, 20]$. In instances with "random" cost, the budget changes within

the range $[0,3]$, initially set to 1 and the magnitude of changes is $\delta \in \{-0.1, 0.1\}$. The results are presented in Table 8.2.

Empirical Analysis

In this section, we compare the performance of three evolutionary and two greedy algorithms in dynamic environments with a variety of frequencies. In Section 8.4.2, we divided the sequence of dynamic changes into four intervals to study how our evolutionary algorithms improve during the optimisation process. In this experiment, we consider all 200 changes as a whole, and we calculate the partial offline error for all the changes. Since smaller benchmarks are considered here, we can study how long does it take for the algorithms to compensate the errors generated at the beginning of the dynamic changes and outperform the others after 200 changes.

Note that the random cost is similar to the cardinality constraint in Section 8.4.2 since it considers each node despite the structure of the graph. In other words, the increase in fitness value occurred by adding a specific node is independent of the rise in the cost. Adding a node that covers lots of other nodes, however, is always expensive when we consider the outdegree cost. Thus, we expect better results from the greedy approaches when the random constraint is considered.

Generally observing the results in Table 2, AdGGA performed worst than GGA in most of the cases. It should be noticed that its inability to trace the best solution is already proven in Section 8.3.2. However, the results confirm that in comparison with the evolutionary approaches, AdGGA's greedy behaviour is still beneficial in the environment that the frequency of changes is high. On the other hand, we have EAMC that has the worst performance among the other iterative algorithms. It does not statistically outperform any of the algorithms in our experiments. Considering the random cost, it achieves a lower mean of partial error only in the smaller benchmark and $\tau = 45000$. While 45000 is still significantly less than the proven theoretical expected time $O(n^3)$, EAMC cannot compete with the other algorithms. The situation becomes a bit better with the outdegree cost. Although it is still unable to outperform the others, the mean of its partial offline error becomes lower than AdGGA's in $\tau = 15000$. The reason for such a poor performance is how EAMC compares solutions to keep them in the population. Despite the fact that its approach guarantees the polynomial size of the population, it increases the chance of storing a bad solution only because no better solution with the same size has been found yet. Hence, it needs more time to prepare its population for the next dynamic change. In the following, we compare the performance of the rest of the algorithms in detail.

Consider the results for smaller benchmark, frb30-15. In instances with $\tau = 100$ and random cost, GGA and AdGGA are significantly better than the other algorithms, and NSGA-II achieves better results than G-SEMO$^{\text{wp}}$. The benefit of the greedy approach in the random cost, do not let NSGA-II dominate GGA before doing 15000

TABLE 8.2: The mean, standard deviation values and statistical tests of the partial offline error for GGA, AdGGA, G-SEMO^{WP}, EAMC, and NSGA-II with elitism, based on the number of evaluations.

	n	r	τ	GGA (1)			AdGGA (2)			G-SEMO^{WP} (3)			EAMC (4)			NSGA-II (5)		
				mean	st	stat	mean	st	stat	mean	st	stat	mean	st	stat	mean	st	stat
frb30-15	450	0.1	100	2.12	0.55	$2^{(+)},3^{(+)},4^{(+)},5^{(+)}$	8.92	1.30	$1^{(-)},3^{(+)},4^{(+)}$	33.87	13.35	$1^{(-)},2^{(-)}$	59.57	24.18	$1^{(-)},2^{(-)},5^{(-)}$	19.78	9.96	$1^{(-)},4^{(+)}$
random	450	0.1	1000	2.12	0.55	$2^{(+)},3^{(+)},4^{(+)}$	8.92	1.30	$1^{(-)},4^{(+)},5^{(-)}$	10.60	3.94	$1^{(-)},4^{(+)},5^{(-)}$	28.16	9.14	$1^{(-)},2^{(-)},3^{(-)},5^{(-)}$	5.27	1.86	$2^{(+)},3^{(+)},4^{(+)}$
	450	0.1	5000	2.12	0.55	$2^{(+)},3^{(+)},4^{(+)}$	8.92	1.30	$1^{(-)},3^{(-)},5^{(-)}$	3.60	1.02	$1^{(-)},2^{(+)},4^{(+)},5^{(-)}$	15.20	4.86	$1^{(-)},3^{(-)},5^{(-)}$	2.15	0.75	$2^{(+)},3^{(+)},4^{(+)}$
	450	0.1	15000	2.12	0.55	$2^{(+)},4^{(+)},5^{(-)}$	8.92	1.30	$1^{(-)},3^{(-)},5^{(-)}$	1.55	0.31	$2^{(+)},4^{(+)}$	10.20	3.37	$1^{(-)},3^{(-)},5^{(-)}$	1.13	0.33	$1^{(+)},2^{(+)},4^{(+)}$
	450	0.1	45000	2.12	0.55	$2^{(+)},3^{(-)},4^{(+)},5^{(-)}$	8.92	1.30	$1^{(-)},3^{(-)},5^{(-)}$	0.67	0.14	$1^{(+)},2^{(+)},4^{(+)}$	6.76	2.29	$1^{(-)},3^{(-)},5^{(-)}$	0.59	0.23	$1^{(+)},2^{(+)},4^{(+)}$
frb30-15	450	20	100	16.66	1.54	$2^{(+)},3^{(+)},4^{(+)},5^{(+)}$	23.83	5.32	$1^{(-)},4^{(+)}$	28.75	3.21	$1^{(-)},4^{(+)}$	43.18	6.57	$1^{(-)},2^{(-)},3^{(-)},5^{(-)}$	28.18	3.62	$1^{(-)},4^{(+)}$
outdegree	450	20	1000	16.66	1.54	$2^{(+)},4^{(+)},5^{(+)}$	23.83	5.32	$1^{(-)},3^{(-)}$	16.36	2.46	$2^{(+)},4^{(+)},5^{(+)}$	32.54	6.88	$1^{(-)},3^{(-)},5^{(-)}$	19.48	2.32	$1^{(-)},3^{(-)},4^{(+)}$
	450	20	5000	16.66	1.54	$2^{(+)},3^{(-)},4^{(+)},5^{(-)}$	23.83	5.32	$1^{(-)},3^{(-)},5^{(-)}$	9.01	2.11	$1^{(+)},2^{(+)},4^{(+)}$	27.69	6.31	$1^{(-)},3^{(-)},5^{(-)}$	12.58	2.56	$1^{(+)},2^{(+)},4^{(+)}$
	450	20	15000	16.66	1.54	$2^{(+)},3^{(-)},5^{(-)}$	23.83	5.32	$1^{(-)},3^{(-)},5^{(-)}$	6.14	1.75	$1^{(+)},2^{(+)},4^{(+)}$	22.75	6.16	$3^{(-)},5^{(-)}$	9.95	2.14	$1^{(+)},2^{(+)},4^{(+)}$
	450	20	45000	16.66	1.54	$2^{(+)},3^{(-)},4^{(+)},5^{(-)}$	23.83	5.32	$1^{(-)},3^{(-)},5^{(-)}$	4.25	1.87	$1^{(+)},2^{(+)},4^{(+)}$	21.71	3.23	$1^{(-)},3^{(-)},5^{(-)}$	7.89	2.03	$1^{(+)},2^{(+)},4^{(+)}$
frb35-17	595	0.1	100	2.28	0.91	$3^{(+)},4^{(+)},5^{(+)}$	6.25	2.61	$3^{(+)},4^{(+)},5^{(+)}$	54.35	21.71	$1^{(-)},2^{(-)}$	100.95	45.88	$1^{(-)},2^{(-)},5^{(-)}$	31.27	13.57	$1^{(-)},2^{(-)},4^{(+)}$
random	595	0.1	1000	2.28	0.91	$2^{(+)},3^{(+)},4^{(+)},5^{(+)}$	6.25	2.61	$1^{(-)},3^{(+)},4^{(+)}$	18.58	8.93	$1^{(-)},2^{(-)},4^{(+)}$	43.73	17.72	$1^{(-)},2^{(-)},3^{(-)},5^{(-)}$	9.49	4.08	$1^{(-)},4^{(+)}$
	595	0.1	5000	2.28	0.91	$2^{(+)},3^{(+)},4^{(+)}$	6.25	2.61	$1^{(-)},4^{(+)},5^{(-)}$	6.32	2.67	$1^{(-)},4^{(+)},5^{(-)}$	22.13	7.55	$1^{(-)},2^{(-)},3^{(-)},5^{(-)}$	3.38	1.04	$2^{(+)},3^{(+)},4^{(+)}$
	595	0.1	15000	2.28	0.91	$2^{(+)},4^{(+)}$	6.25	2.61	$1^{(-)},3^{(-)},5^{(-)}$	2.36	0.82	$2^{(+)},4^{(+)},5^{(-)}$	13.73	4.21	$1^{(-)},3^{(-)},5^{(-)}$	1.40	0.43	$2^{(+)},3^{(+)},4^{(+)}$
	595	0.1	45000	2.28	0.91	$2^{(+)},3^{(-)},4^{(+)},5^{(-)}$	6.25	2.61	$1^{(-)},3^{(-)},5^{(-)}$	0.73	0.25	$1^{(+)},2^{(+)},4^{(+)}$	8.53	2.23	$1^{(-)},3^{(-)},5^{(-)}$	0.62	0.25	$1^{(+)},2^{(+)},4^{(+)}$
frb35-17	595	20	100	16.00	1.08	$2^{(+)},3^{(+)},4^{(+)},5^{(+)}$	35.24	10.69	$1^{(-)},4^{(+)}$	38.48	3.71	$1^{(-)},4^{(+)}$	56.11	6.40	$1^{(-)},2^{(-)},3^{(-)},5^{(-)}$	37.82	4.50	$1^{(-)},4^{(+)}$
outdegree	595	20	1000	16.00	1.08	$2^{(+)},3^{(+)},4^{(+)},5^{(+)}$	35.24	10.69	$1^{(-)},3^{(-)}$	21.25	2.73	$1^{(-)},2^{(+)},4^{(+)}$	46.23	7.88	$1^{(-)},3^{(-)},5^{(-)}$	25.41	3.45	$1^{(-)},4^{(+)}$
	595	20	5000	16.00	1.08	$2^{(+)},3^{(-)},4^{(+)}$	35.24	10.69	$1^{(-)},3^{(-)},5^{(-)}$	9.77	2.42	$1^{(+)},2^{(+)},4^{(+)},5^{(+)}$	36.30	8.20	$1^{(-)},3^{(-)},5^{(-)}$	16.07	3.23	$2^{(+)},3^{(-)},4^{(+)}$
	595	20	15000	16.00	1.08	$2^{(+)},3^{(-)},4^{(+)}$	35.24	10.69	$1^{(-)},3^{(-)},5^{(-)}$	4.93	2.40	$1^{(+)},2^{(+)},4^{(+)},5^{(+)}$	31.93	6.15	$1^{(-)},3^{(-)},5^{(-)}$	11.96	2.86	$2^{(+)},3^{(-)},4^{(+)}$
	595	20	45000	16.00	1.08	$2^{(+)},3^{(-)},4^{(+)}$	35.24	10.69	$1^{(-)},3^{(-)},5^{(-)}$	2.12	1.86	$1^{(+)},2^{(+)},4^{(+)}$	27.00	4.74	$1^{(-)},3^{(-)},5^{(-)}$	8.19	2.51	$2^{(+)},4^{(+)}$

evaluations. For G-SEMO^{wp}, it is more challenging to get better than greedy algorithms. It requires 15000 evaluations before each change to perform as good as NSGA-II, and 45000 evaluations to outperform GGA. However, the situation is different when the cost function is calculated based on the outdegree. $\tau = 5000$ is enough for G-SEMO^{wp} and NSGA-II to outperform the greedy algorithms.

For the more complicated benchmark, frb35-17, almost the same results hold. However, it gets harder for our evolutionary algorithms to beat greedy algorithms. Under the random cost, AdGGA outperforms NSGA-II, EAMC and G-SEMO^{wp}, when $\tau = 100$ and GGA remains unbeatable until $\tau = 45000$. GGA also gains better results with outdegree cost. It outperforms other algorithms in $\tau = 100$ and 1000 and loses only to G-SEMO afterward. These results are reasonable since the size of solutions increases to 595 bits in the frb35-17 benchmark, which increases the expected time for each bit to be flipped by an evolutionary algorithm. Consequently, our algorithms need more time to find better solutions. However, the results show that NSGA-II seems to have more difficulties in dealing with the bigger size than G-SEMO^{wp}.

The other fact extracted from the results is that NSGA-II has a better performance than G-SEMO^{wp} when we have random cost. However, the results of G-SEMO^{wp} are better at outdegree cost. This pattern can be observed in both benchmarks. That is the result of the population size and how each algorithm handles the replacement of a new solution. While G-SEMO^{wp} can store a valuable solution for each possible cost, the NSGA-II concentrates on producing a well-distributed population. Consider the cases that there are a lot of non-dominated solutions with cost values close to the constraint. G-SEMO^{wp} can keep them all, which is an advantage if the next dynamic change does not affect the constraint significantly. However, because of the distribution factor, NSGA-II minimises the number of solutions that are close to each other. These situations rarely happen with random constraint, since, as discussed previously, it is similar to the cardinality constraint and the possible values for the cost of the solutions are more evenly distributed.

8.5 Conclusions

Many real-world problems can be modelled as submodular functions and have problems with dynamically changing constraints. We have contributed to the area of submodular optimisation with dynamic constraints. Key to the investigations have been the adaptability of algorithms when constraints change. We have shown that an adaptive version of the generalised greedy algorithm frequently considered in submodular constrained optimisation is not able to maintain a ϕ-approximation. Furthermore, we have pointed out that the population-based G-SEMO algorithm is able to cater for and recompute ϕ-approximations for related constraints bounds in an efficient way. We challenged G-SEMO and greedy approaches against EAMC as a recently introduced algorithm with polynomial expected running time, and NSGA-II

as an advanced multi-objective algorithm in practice. Our experimental results confirm the advantage of G-SEMO and NSGA-II over the considered greedy approaches on important real-world problems. Furthermore, they show that evolutionary algorithms are able to significantly improve their performance over time when dealing with dynamic changes.

Chapter 9

Conclusion

Bio-inspired strategies are general-purpose optimisation techniques that are inspired by nature. These algorithms can find high-quality solutions for complex problems without much knowledge about the search space. In this thesis, we contributed to the theoretical and practical understanding of bio-inspired algorithms dealing with complex problems, as well as problems with dynamically changing constraints.

In Part I of the thesis, we introduced the bio-inspired algorithms to be considered in the subsequent chapters, followed by a description of theoretical tools designed to analyse these algorithms (Chapter 2). Next, in Chapter 3, we explained combinatorial optimisation and presented a detailed definition of well-known combinatorial problems that we studied in this thesis. Evolutionary algorithms and other bio-inspired algorithms have been widely applied to dynamic problems. In Chapter 4, the final chapter of Part I, we provided an overview of major theoretical developments in the area of runtime analysis for these problems. We reviewed recent theoretical studies of evolutionary algorithms and ant colony optimisation for problems where the objective functions or the constraints change over time.

Part II of the thesis was dedicated to the investigation of static combinatorial problems. In Chapter 5, we considered a single- and multi-objective version of the minimum spanning tree (MST) problem, as well as introducing a biased mutation. A biased mutation puts more emphasis on the selection of low-ranking edges in terms of low domination number. We presented example graphs in which the biased mutation significantly sped up the expected runtime until (Pareto-)optimal solutions were found. On the other hand, we demonstrated that bias can lead to exponential runtime if "heavy" edges are necessarily part of an optimal solution. However, on general graphs in the single-objective setting, we showed that a combined mutation operator, which decides for unbiased or biased edge selection in each step with equal probability, exhibits a polynomial upper bound – as unbiased mutation – in the worst case and benefits from bias if the circumstances are favourable.

From a theoretical viewpoint, the literature extensively investigates the performance of baseline evolutionary algorithms on linear problems, while the theoretical analysis of non-linear problems is still far behind. In Chapter 6, variations of the packing while travelling (PWT) problem – also known as the non-linear knapsack problem – were studied in an attempt to analyse the performance of EAs to solve non-linear problems from a theoretical perspective. We investigated PWT for two cities and n items with correlated weights and profits, using single-objective and multi-objective algorithms. Our results show that RLS_swap, which differs from the classical RLS in its ability to swap two bits in one iteration, finds the optimal solution in $O(n^3)$ expected time. We also studied an enhanced version of G-SEMO, which uses a specific selection operator to deal with exponential population size, and proved that it finds the Pareto front in the same asymptotic expected time. In the case of uniform weights, $(1+1)$ EA is able to find the optimal solution in expected time $O(n^2 \log(\max\{n, p_{\max}\}))$, where $p_{\max}$ is the largest profit of the given items. We also performed an experimental analysis to complement our theoretical investigations and provide additional insights into the runtime behaviour of our algorithms solving PWT.

In the Part III, we studied the performance of bio-inspired algorithms to solve dynamic problems. In Chapter 7, we studied single- and multi-objective baseline evolutionary algorithms for the classical knapsack problem where the capacity of the knapsack varies over time. We established different benchmark scenarios, where the capacity changes every τ iterations according to a uniform or normal distribution. Our experimental investigations analysed the behaviour of our algorithms in terms of the magnitude of changes determined by the parameters of the chosen distribution, the frequency determined by τ, and the class of knapsack instances under consideration. Our results showed that the multi-objective approaches using a population that caters for dynamic changes have a clear advantage in many benchmark scenarios when the frequency of changes is not too high. Furthermore, we demonstrated that the distribution handling techniques in advanced algorithms such as NSGA-II and SPEA2 do not necessarily result in better performance, and even prevent these algorithms from finding good quality solutions in comparison with simple multi-objective approaches.

Finally, in Chapter 8, we considered the subset selection problem for function f with a constraint bound B that changes over time. We pointed out that adaptive variants of greedy approaches commonly used in the area of submodular optimisation are not able to maintain their approximation quality in dynamic environments. Investigating the G-SEMO Pareto optimisation approach, we showed that this algorithm efficiently computes a $\phi = (\alpha_f/2)(1 - \frac{1}{e^{\alpha_f}})$-approximation, where α_f is the submodularity ratio of f, for each possible constraint bound $b \leq B$. Furthermore, we showed that G-SEMO is able to adapt its set of solutions quickly in the case that B increases. Our experimental investigations for the influence maximisation in social networks

showed the advantage of G-SEMO over generalised greedy algorithms. We also considered EAMC, a new evolutionary algorithm with polynomial expected time guaranteed to maintain a ϕ approximation ratio, and NSGA-II (an advanced multi-objective optimisation algorithm), with the aim of demonstrating their challenges in optimising the maximum coverage problem. Our empirical analysis showed that, within the same number of evaluations, G-SEMO is able to outperform NSGA-II under linear constraints, while EAMC performs significantly worse than all considered algorithms in most cases.

In this thesis, we tried to provide more insight into the performance of bio-inspired algorithms from both theoretical and practical points of views. While we contributed to the understanding of the basic working principles of bio-inspired algorithms, analysing their ability to solve complex problems remains highly open. In the case of using knowledge-based mutation to solve MST problem, we still want to extend our theories to general graphs and support them with practical results. Moreover, we aim to extend our results for the PWT to more than two cities, with correlated items and items with uniform weights. However, cases in which items are uncorrelated, or are correlated separately in each city, might prove as hard to solve as the general case. Furthermore, if we were to extend our use of bio-inspired algorithms to dynamic environments, we might find that uncertainties often change over time and are therefore dynamic. It would be interesting to analyse the ability of evolutionary algorithms to solve problems where uncertainties change over time. For future research, it would also be interesting to examine environments that are both dynamic and stochastic, as many real-world problems have both properties at the same time.

Bibliography

[AB02] Réka Albert and Albert-László Barabási. "Statistical mechanics of complex networks". In: *Reviews of modern physics* 74.1 (2002), p. 47.

[AD18] Denis Antipov and Benjamin Doerr. "Precise Runtime Analysis for Plateaus". In: *Parallel Problem Solving from Nature - PPSN XV - 15th International Conference, Coimbra, Portugal, September 8-12, 2018, Proceedings, Part II*. Ed. by Anne Auger et al. Vol. 11102. Lecture Notes in Computer Science. Springer, 2018, pp. 117–128.

[AHN18] Maria Yaneli Ameca-Alducin, Maryam Hasani-Shoreh, and Frank Neumann. "On the Use of Repair Methods in Differential Evolution for Dynamic Constrained Optimization". In: *Applications of Evolutionary Computation, Proceedings*. 2018, pp. 832–847.

[BBM12] Nicola Barbieri, Francesco Bonchi, and Giuseppe Manco. "Topic-aware social influence propagation models". In: *IEEE Conference on Data Mining*. IEEE Computer Society. 2012, pp. 81–90.

[BGN19] Jakob Bossek, Christian Grimme, and Frank Neumann. "On the benefits of biased edge-exchange mutation for the multi-criteria spanning tree problem". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*. Ed. by Anne Auger and Thomas Stützle. ACM, 2019, pp. 516–523.

[Bia+20] Chao Bian et al. "An Efficient Evolutionary Algorithm for Subset Selection with General Cost Constraints". In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 2020, pp. 3267–3274.

[BMB13] Mohammad Reza Bonyadi, Zbigniew Michalewicz, and Luigi Barone. "The travelling thief problem: The first step in the transition from theoretical problems to realistic problems". In: *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2013, Cancun, Mexico, June 20-23, 2013*. IEEE, 2013, pp. 1037–1044.

[BÖ14] Adil Baykasoglu and Fehmi Burcin Özsoydan. "An improved firefly algorithm for solving dynamic multidimensional knapsack problems". In: *Expert Syst. Appl.* 41.8 (2014), pp. 3712–3725.

[BÖ17] Adil Baykasoglu and Fehmi Burcin Özsoydan. "Evolutionary and population-based methods versus constructive search strategies in dynamic combinatorial optimization". In: *Inf. Sci.* 420 (2017), pp. 159–183.

[Bon+19] Mohammad Reza Bonyadi et al. "Evolutionary Computation for Multicomponent Problems: Opportunities and Future Directions". In: *Optimization in Industry: Present Practices and Future Scopes*. Ed. by Shubhabrata Datta and J. Paulo Davim. Cham: Springer International Publishing, 2019, pp. 13–30.

[Bos+19] Jakob Bossek et al. "Runtime analysis of randomized search heuristics for dynamic graph coloring". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*. Ed. by Anne Auger and Thomas Stützle. ACM, 2019, pp. 1443–1451.

[Bro89] Andrei Z. Broder. "Generating Random Spanning Trees". In: *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*. IEEE Computer Society, 1989, pp. 442–447.

[CC84] Michele Conforti and Gérard Cornuéjols. "Submodular set functions, matroids and the greedy algorithm: Tight worst-case bounds and some generalizations of the Rado-Edmonds theorem". In: *Discrete Applied Mathematics* 7.3 (1984), pp. 251–274.

[CF09] Gregory W. Corder and Dale I. Foreman. *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*. Wiley, 2009. ISBN: 047045461X.

[Cor+16] Dogan Corus et al. "A parameterised complexity analysis of bi-level optimisation with evolutionary algorithms". In: *Evolutionary computation* 24.1 (2016), pp. 183–203.

[CWM11] Raymond Chiong, Thomas Weise, and Zbigniew Michalewicz. *Variants of Evolutionary Algorithms for Real-World Applications*. Springer Publishing Company, Incorporated, 2011.

[DDE15] Benjamin Doerr, Carola Doerr, and Franziska Ebel. "From black-box complexity to designing new genetic algorithms". In: *Theor. Comput. Sci.* 567 (2015), pp. 87–104.

[Deb+02] Kalyanmoy Deb et al. "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE Trans. Evolutionary Computation* 6.2 (2002), pp. 182–197.

[Deb01] Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. New York, NY, USA: John Wiley & Sons, Inc., 2001.

[DG13] Benjamin Doerr and Leslie Ann Goldberg. "Adaptive Drift Analysis". In: *Algorithmica* 65.1 (2013), pp. 224–250.

[DHK11] Benjamin Doerr, Edda Happ, and Christian Klein. "Tight Analysis of the (1+1)-EA for the Single Source Shortest Path Problem". In: *Evolutionary Computation* 19.4 (2011), pp. 673–691.

[DHN06] Benjamin Doerr, Nils Hebbinghaus, and Frank Neumann. "Speeding Up Evolutionary Algorithms Through Restricted Mutation Operators". In: *Parallel Problem Solving from Nature - PPSN IX*. Ed. by Thomas Philip Runarsson et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 978–987.

[DHN07] Benjamin Doerr, Nils Hebbinghaus, and Frank Neumann. "Speeding Up Evolutionary Algorithms Through Asymmetric Mutation Operators". In: *Evolutionary Computation* 15.4 (Dec. 2007), pp. 401–410.

[DJW02] Stefan Droste, Thomas Jansen, and Ingo Wegener. "On the analysis of the (1+1) evolutionary algorithm". In: *Theor. Comput. Sci.* 276.1-2 (2002), pp. 51–81.

[DJW12] Benjamin Doerr, Daniel Johannsen, and Carola Winzen. "Multiplicative drift analysis". In: *Algorithmica* 64.4 (2012), pp. 673–697.

[DN11] Juan José Durillo and Antonio J. Nebro. "jMetal: A Java framework for multi-objective optimization". In: *Advances in Engineering Software* 42.10 (2011), pp. 760–771.

[DN20] Benjamin Doerr and Frank Neumann. *Theory of Evolutionary Computation—Recent Developments in Discrete Optimization*. 2020.

[Doe20] Benjamin Doerr. "Probabilistic Tools for the Analysis of Randomized Optimization Heuristics". In: *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*. Ed. by Benjamin Doerr and Frank Neumann. Cham: Springer International Publishing, 2020, pp. 1–87. ISBN: 978-3-030-29414-4.

[Dro02] 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*. Vol. 1. 2002, pp. 55–60.

[DS04] Marco Dorigo and Thomas Stützle. *Ant colony optimization*. MIT Press, 2004.

[DW18] Carola Doerr and Markus Wagner. "Simple on-the-fly parameter selection mechanisms for two classical discrete black-box optimization benchmark problems". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, Kyoto, Japan, July 15-19, 2018*. Ed. by Hernán E. Aguirre and Keiki Takadama. ACM, 2018, pp. 943–950.

[ER59] P Erdős and A Rényi. "On random graphs I". In: *Publicationes Mathematicae Debrecen* 6 (1959), pp. 290–297.

[ES15] A. E. Eiben and James E. Smith. *Introduction to Evolutionary Computing, Second Edition*. Natural Computing Series. Springer, 2015.

[FF95] Carlos M. Fonseca and Peter J. Fleming. "An Overview of Evolutionary Algorithms in Multiobjective Optimization". In: *Evol. Comput.* 3.1 (1995), pp. 1–16.

[FQW18] Tobias Friedrich, Francesco Quinzan, and Markus Wagner. "Escaping Large Deceptive Basins of Attraction with Heavy-tailed Mutation Operators". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '18. New York, NY, USA: ACM, 2018, pp. 293–300.

[Fri+10] Tobias Friedrich et al. "Approximating Covering Problems by Randomized Search Heuristics Using Multi-Objective Models". In: *Evolutionary Computation* 18.4 (2010), pp. 617–633.

[Fri+17] Tobias Friedrich et al. "Analysis of the (1+1) EA on Subclasses of Linear Functions under Uniform and Linear Constraints". In: *Proceedings of the 14th ACM/SIGEVO Conference on Foundations of Genetic Algorithms, FOGA 2017, Copenhagen, Denmark, January 12-15, 2017*. Ed. by Christian Igel, Dirk Sudholt, and Carsten Witt. ACM, 2017, pp. 45–54.

[Fri+18] Tobias Friedrich et al. "Heavy-Tailed Mutation Operators in Single-Objective Combinatorial Optimization". In: *Parallel Problem Solving from Nature – PPSN XV*. Ed. by Anne Auger et al. Cham: Springer International Publishing, 2018, pp. 134–145.

[Gie03] Oliver Giel. "Expected runtimes of a simple multi-objective evolutionary algorithm". In: *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.* Vol. 3. IEEE. 2003, pp. 1918–1925.

[GK16] Christian Gießen and Timo Kötzing. "Robustness of populations in stochastic environments". In: *Algorithmica* 75.3 (2016), pp. 462–489.

[HL12] Tad Hogg and Kristina Lerman. "Social Dynamics of Digg". In: *EPJ Data Science* 1.1 (2012), p. 5.

[HY01] Jun He and Xin Yao. "Drift analysis and average time complexity of evolutionary algorithms". In: *Artificial intelligence* 127.1 (2001), pp. 57–85.

[IB13] Rishabh K. Iyer and Jeff A. Bilmes. "Submodular Optimization with Submodular Cover and Submodular Knapsack Constraints". In: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*. Ed. by Christopher J. C. Burges et al. 2013, pp. 2436–2444.

[Jan13] Thomas Jansen. *Analyzing Evolutionary Algorithms - The Computer Science Perspective*. Natural Computing Series. Springer, 2013.

[JOZ13] Thomas Jansen, Pietro Simone Oliveto, and Christine Zarges. "Approximating vertex cover using edge-based representations". In: *Foundations of Genetic Algorithms XII, FOGA '13, Adelaide, SA, Australia, January 16-20, 2013*. Ed. by Frank Neumann and Kenneth A. De Jong. ACM, 2013, pp. 87–96.

[JS10] Thomas Jansen and Dirk Sudholt. "Analysis of an Asymmetric Mutation Operator". In: *Evolutionary Computation* 18.1 (Mar. 2010), pp. 1–26.

[JZ14] Thomas Jansen and Christine Zarges. "Evolutionary algorithms and artificial immune systems on a bi-stable dynamic optimisation problem".

In: *Genetic and Evolutionary Computation Conference, GECCO '14, Vancouver, BC, Canada, July 12-16, 2014*. Ed. by Dirk V. Arnold. ACM, 2014, pp. 975–982.

[Kan87] Mikio Kano. "Maximum and k-th maximal spanning trees of a weighted graph". In: *Combinatorica* 7.2 (1987), pp. 205–214.

[KC01] J.D. Knowles and D.W. Corne. "A Comparison of Encodings and Algorithms for Multiobjective Minimum Spanning Tree Problems". In: *Evolutionary Computation* 1 (2001), pp. 544–551.

[KE95] James Kennedy and Russell Eberhart. "Particle swarm optimization". In: *Proceedings of International Conference on Neural Networks (ICNN'95), Perth, WA, Australia, November 27 - December 1, 1995*. IEEE, 1995, pp. 1942–1948.

[KG05] Andreas Krause and Carlos Guestrin. *A note on the budgeted maximization of submodular functions*. Carnegie Mellon University. Center for Automated Learning and Discovery, 2005.

[KKT15] David Kempe, Jon M. Kleinberg, and Éva Tardos. "Maximizing the Spread of Influence through a Social Network". In: *Theory of Computing* 11 (2015), pp. 105–147.

[KLW15] Timo Kötzing, Andrei Lissovoi, and Carsten Witt. "(1+1) EA on Generalized Dynamic OneMax". In: *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII, Aberystwyth, United Kingdom, January 17 - 20, 2015*. Ed. by Jun He et al. ACM, 2015, pp. 40–51.

[KM12] Timo Kötzing and Hendrik Molter. "ACO Beats EA on a Dynamic Pseudo-Boolean Function". In: *Parallel Problem Solving from Nature - PPSN XII - 12th International Conference, Taormina, Italy, September 1-5, 2012, Proceedings, Part I*. Ed. by Carlos A. Coello Coello et al. Vol. 7491. Lecture Notes in Computer Science. Springer, 2012, pp. 113–122.

[KMN99] Samir Khuller, Anna Moss, and Joseph Naor. "The Budgeted Maximum Coverage Problem". In: *Inf. Process. Lett.* 70.1 (1999), pp. 39–45.

[KN13] Stefan Kratsch and Frank Neumann. "Fixed-Parameter Evolutionary Algorithms and the Vertex Cover Problem". In: *Algorithmica* 65.4 (2013), pp. 754–771.

[KPP04] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004. ISBN: 978-3-540-40286-2.

[Kru56] Joseph B. Kruskal. "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem". In: *Proceedings of the American Mathematical Society* 7.1 (1956), pp. 48–50.

[Lau+02] Marco Laumanns et al. "Running Time Analysis of Multi-objective Evolutionary Algorithms on a Simple Discrete Optimization Problem". In: *Parallel Problem Solving from Nature - PPSN VII, 7th International Conference, Granada, Spain, September 7-11, 2002, Proceedings*. Ed. by Juan Julián

Merelo Guervós et al. Vol. 2439. Lecture Notes in Computer Science. Springer, 2002, pp. 44–53.

[LB10] Hui Lin and Jeff A. Bilmes. "Multi-document Summarization via Budgeted Maximization of Submodular Functions". In: *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 2-4, 2010, Los Angeles, California, USA*. The Association for Computational Linguistics, 2010, pp. 912–920.

[Len20] Johannes Lengler. "Drift Analysis". In: *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*. Ed. by Benjamin Doerr and Frank Neumann. Cham: Springer International Publishing, 2020, pp. 89–131. ISBN: 978-3-030-29414-4.

[LW13] Per Kristian Lehre and Carsten Witt. "General drift analysis with tail bounds". In: *arXiv preprint arXiv:1307.2559* (2013).

[LW15] Andrei Lissovoi and Carsten Witt. "Runtime analysis of ant colony optimization on dynamic shortest path problems". In: *Theor. Comput. Sci.* 561 (2015), pp. 73–85.

[LW16] Andrei Lissovoi and Carsten Witt. "MMAS Versus Population-Based EA on a Family of Dynamic Fitness Functions". In: *Algorithmica* 75.3 (2016), pp. 554–576.

[LW17] Andrei Lissovoi and Carsten Witt. "A Runtime Analysis of Parallel Evolutionary Algorithms in Dynamic Optimization". In: *Algorithmica* 78.2 (2017), pp. 641–659.

[MA94] Zbigniew Michalewicz and Jaroslaw Arabas. "Genetic Algorithms for the 0/1 Knapsack Problem". In: *Methodologies for Intelligent Systems, 8th International Symposium, ISMIS '94, Charlotte, North Carolina, USA, October 16-19, 1994, Proceedings*. Ed. by Zbigniew W. Ras and Maria Zemankova. Vol. 869. Lecture Notes in Computer Science. Springer, 1994, pp. 134–143.

[MI95] Tadahiko Murata and Hisao Ishibuchi. "MOGA: multi-objective genetic algorithms". In: *IEEE international conference on evolutionary computation*. Vol. 1. 1995, pp. 289–294.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Page 61. Cambridge University Press, 1995.

[MST15] Aleksander Madry, Damian Straszak, and Jakub Tarnawski. "Fast Generation of Random Spanning Trees and the Effective Resistance Metric". In: *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*. Ed. by Piotr Indyk. SIAM, 2015, pp. 2019–2036.

[Müh92] Heinz Mühlenbein. "How Genetic Algorithms Really Work: Mutation and Hillclimbing". In: *Parallel Problem Solving from Nature 2, PPSN-II, Brussels, Belgium, September 28-30, 1992*. Ed. by Reinhard Männer and Bernard Manderick. Elsevier, 1992, pp. 15–26.

[NB03] Siegfried Nijssen and Thomas Bäck. "An analysis of the behavior of simplified evolutionary algorithms on trap functions". In: *IEEE Trans. Evolutionary Computation* 7.1 (2003), pp. 11–22.

[Neu+18] Frank Neumann et al. "A Fully Polynomial Time Approximation Scheme for Packing While Traveling". In: *Algorithmic Aspects of Cloud Computing - 4th International Symposium, ALGOCLOUD 2018, Helsinki, Finland, August 20-21, 2018, Revised Selected Papers*. Ed. by Yann Disser and Vassilios S. Verykios. Vol. 11409. Lecture Notes in Computer Science. Springer, 2018, pp. 59–72.

[NPR20] Frank Neumann, Mojgan Pourhassan, and Vahid Roostapour. "Analysis of Evolutionary Algorithms in Dynamic and Stochastic Environments". In: *Theory of Evolutionary Computation*. Springer, 2020, pp. 323–357.

[NS18] Frank Neumann and Andrew M. Sutton. "Runtime Analysis of Evolutionary Algorithms for the Knapsack Problem with Favorably Correlated Weights". In: *Parallel Problem Solving from Nature - PPSN XV - 15th International Conference, Coimbra, Portugal, September 8-12, 2018, Proceedings, Part II*. Ed. by Anne Auger et al. Vol. 11102. Lecture Notes in Computer Science. Springer, 2018, pp. 141–152.

[NW06] Frank Neumann and Ingo Wegener. "Minimum spanning trees made easier via multi-objective optimization". In: *Natural Computing* 5.3 (2006), pp. 305–319.

[NW07] Frank Neumann and Ingo Wegener. "Randomized local search, evolutionary algorithms, and the minimum spanning tree problem". In: *Theoretical Computer Science* 378.1 (2007), pp. 32–40.

[NW10a] Frank Neumann and Carsten Witt. "Ant Colony Optimization and the minimum spanning tree problem". In: *Theoretical Computer Science* 411.25 (2010), pp. 2406–2413.

[NW10b] Frank Neumann and Carsten Witt. *Bioinspired Computation in Combinatorial Optimization*. Natural Computing Series. Springer, 2010.

[NW15] Frank Neumann and Carsten Witt. "On the Runtime of Randomized Local Search and Simple Evolutionary Algorithms for Dynamic Makespan Scheduling". In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*. Ed. by Qiang Yang and Michael J. Wooldridge. AAAI Press, 2015, pp. 3742–3748.

[NW81] George L Nemhauser and Laurence A Wolsey. "Maximizing submodular set functions: formulations and analysis of algorithms". In: *North-Holland Mathematics Studies*. Vol. 59. Elsevier, 1981, pp. 279–301.

[NY12] T.T. Nguyen and X. Yao. "Continuous Dynamic Constrained Optimization: The Challenges". In: *IEEE Transactions on Evolutionary Computation* 16.6 (2012), pp. 769–786. ISSN: 1089-778X. DOI: 10.1109/TEVC.2011.2180533.

[NYB12] Trung Thanh Nguyen, Shengxiang Yang, and Jürgen Branke. "Evolutionary dynamic optimization: A survey of the state of the art". In: *Swarm and Evolutionary Computation* 6 (2012), pp. 1–24.

[OHY09] Pietro Simone Oliveto, Jun He, and Xin Yao. "Analysis of the (1+1) -EA for Finding Approximate Solutions to Vertex Cover Problems". In: *IEEE Trans. Evolutionary Computation* 13.5 (2009), pp. 1006–1029.

[PGN15] Mojgan Pourhassan, Wanru Gao, and Frank Neumann. "Maintaining 2-Approximations for the Dynamic Vertex Cover Problem Using Evolutionary Algorithms". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2015, pp. 903–910.

[PN15] Sergey Polyakovskiy and Frank Neumann. "Packing While Traveling: Mixed Integer Programming for a Class of Nonlinear Knapsack Problems". In: *Integration of AI and OR Techniques in Constraint Programming - 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings*. Ed. by Laurent Michel. Vol. 9075. Lecture Notes in Computer Science. Springer, 2015, pp. 332–346.

[PN17] Sergey Polyakovskiy and Frank Neumann. "The Packing While Traveling Problem". In: *European Journal of Operational Research* 258.2 (2017), pp. 424–439.

[PN18] Mojgan Pourhassan and Frank Neumann. "Theoretical Analysis of Local Search and Simple Evolutionary Algorithms for the Generalized Travelling Salesperson Problem". In: *Evolutionary computation* (2018), pp. 1–34.

[Pol+14] Sergey Polyakovskiy et al. "A comprehensive benchmark set and heuristics for the traveling thief problem". In: *Proceedings of Conference on Genetic and Evolutionary Computation*. ACM. 2014, pp. 477–484.

[Pop14] Elena Popovici. "Anne Auger and Benjamin Doerr (eds): Theory of randomized search heuristics: foundations and recent developments - World Scientific (2011), 359 pp". In: *Genetic Programming and Evolvable Machines* 15.1 (2014), pp. 111–112.

[PRN19] Mojgan Pourhassan, Vahid Roostapour, and Frank Neumann. "Runtime analysis of RLS and (1 + 1) EA for the dynamic weighted vertex cover problem". In: *Theoretical Computer Science* (2019). ISSN: 0304-3975.

[PS82] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982. ISBN: 0-13-152462-3.

[PSN16] Mojgan Pourhassan, Feng Shi, and Frank Neumann. "Parameterized Analysis of Multi-objective Evolutionary Algorithms and the Weighted Vertex Cover Problem". In: *Parallel Problem Solving from Nature - PPSN XIV - 14th International Conference, Edinburgh, UK, September 17-21, 2016, Proceedings*. Ed. by Julia Handl et al. Vol. 9921. Lecture Notes in Computer Science. Springer, 2016, pp. 729–739.

[QBF20] Chao Qian, Chao Bian, and Chao Feng. "Subset Selection by Pareto Optimization with Recombination". In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 2020, pp. 2408–2415.

[Qia+17] Chao Qian et al. "On Subset Selection with General Cost Constraints". In: *International Joint Conference on Artificial Intelligence, IJCAI 2017*. 2017, pp. 2613–2619.

[RBN20] Vahid Roostapour, Jakob Bossek, and Frank Neumann. "Runtime Analysis of Evolutionary Algorithms with Biased Mutation for the Multi-Objective Minimum Spanning Tree Problem". In: *arXiv preprint arXiv:2004.10424* (2020).

[RKD17] Pratyusha Rakshit, Amit Konar, and Swagatam Das. "Noisy evolutionary optimization algorithms - A comprehensive survey". In: *Swarm and Evolutionary Computation* 33 (2017), pp. 18–45.

[RKJ06] Günther R. Raidl, Gabriele Koller, and Bryant A. Julstrom. "Biased Mutation Operators for Subgraph-Selection Problems". In: *IEEE Trans. Evolutionary Computation* 10.2 (2006), pp. 145–156.

[RNN18] Vahid Roostapour, Aneta Neumann, and Frank Neumann. "On the Performance of Baseline Evolutionary Algorithms on the Dynamic Knapsack Problem". In: *Parallel Problem Solving from Nature - PPSN XV - 15th International Conference, Coimbra, Portugal, September 8-12, 2018, Proceedings, Part I*. Vol. 11101. Lecture Notes in Computer Science. Springer, 2018, pp. 158–169.

[RNN20] Vahid Roostapour, Aneta Neumann, and Frank Neumann. "Evolutionary Multi-Objective Optimization for the Dynamic Knapsack Problem". In: *arXiv preprint arXiv:2004.12574* (2020).

[Roo+19] Vahid Roostapour et al. "Pareto Optimization for Subset Selection with Dynamic Cost Constraints". In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019.* AAAI Press, 2019, pp. 2354–2361.

[Roo+20] Vahid Roostapour et al. "Pareto Optimization for Subset Selection with Dynamic Cost Constraints. CoRR abs/1811.07806v2 (2020)". In: *arXiv preprint 1811.07806v2* (2020).

[RPN19] Vahid Roostapour, Mojgan Pourhassan, and Frank Neumann. "Analysis of baseline evolutionary algorithms for the packing while travelling problem". In: *Proceedings of the 15th ACM/SIGEVO Conference on Foundations of Genetic Algorithms, FOGA 2019, Potsdam, Germany, August 27-29, 2019*. Ed. by Tobias Friedrich, Carola Doerr, and Dirk V. Arnold. ACM, 2019, pp. 124–132.

[RT89] Keith W. Ross and Danny H. K. Tsang. "The stochastic knapsack problem". In: *IEEE Trans. Communications* 37.7 (1989), pp. 740–747.

[Sah75] Sartaj Sahni. "Approximate Algorithms for the 0/1 Knapsack Problem". In: *J. ACM* 22.1 (1975), pp. 115–124.

[Sch85] J. David Schaffer. "Multiple Objective Optimization with Vector Evaluated Genetic Algorithms". In: *Proceedings of the 1st International Conference on Genetic Algorithms, Pittsburgh, PA, USA, July 1985*. Ed. by John J. Grefenstette. Lawrence Erlbaum Associates, 1985, pp. 93–100.

[SD94] Nidamarthi Srinivas and Kalyanmoy Deb. "Muiltiobjective optimization using nondominated sorting in genetic algorithms". In: *Evolutionary computation* 2.3 (1994), pp. 221–248.

[SH00] Thomas Stützle and Holger H. Hoos. "MAX-MIN Ant System". In: *Future Generation Comp. Syst.* 16.8 (2000), pp. 889–914.

[Shi+19] Feng Shi et al. "Reoptimization Time Analysis of Evolutionary Algorithms on Linear Functions Under Dynamic Uniform Constraints". In: *Algorithmica* 81.2 (2019), pp. 828–857.

[SK18] Kevin Sim and Paul Kaufmann. *Applications of Evolutionary Computation: 21st International Conference, EvoApplications 2018, Parma, Italy, April 4-6, 2018, Proceedings*. Vol. 10784. Springer, 2018.

[SN12] Andrew M. Sutton and Frank Neumann. "A Parameterized Runtime Analysis of Simple Evolutionary Algorithms for Makespan Scheduling". In: *Parallel Problem Solving from Nature - PPSN XII - 12th International Conference, Taormina, Italy, September 1-5, 2012, Proceedings, Part I*. Ed. by Carlos A. Coello Coello et al. Vol. 7491. Lecture Notes in Computer Science. Springer, 2012, pp. 52–61.

[ST12] Dirk Sudholt and Christian Thyssen. "Running time analysis of Ant Colony Optimization for shortest path problems". In: *J. Discrete Algorithms* 10 (2012), pp. 165–180.

[STW04] Jens Scharnow, Karsten Tinnefeld, and Ingo Wegener. "The analysis of evolutionary algorithms on sorting and shortest paths problems". In: *J. Math. Model. Algorithms* 3.4 (2004), pp. 349–366.

[UU09] Sima Uyar and H. Turgut Uyar. "A Critical Look at Dynamic Multidimensional Knapsack Problem Generation". In: *Applications of Evolutionary Computing, EvoWorkshops 2009: EvoCOMNET, EvoENVIRONMENT, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoINTERACTION, EvoMUSART, EvoNUM, EvoSTOC, EvoTRANSLOG, Tübingen, Germany, April 15-17, 2009. Proceedings*. Ed. by Mario Giacobini et al. Vol. 5484. Lecture Notes in Computer Science. Springer, 2009, pp. 762–767.

[Von10] Jan Vondrák. "Submodularity and curvature: The optimal algorithm". In: *RIMS Kôkyûroku Bessatsu* B23 (2010), pp. 253–266.

[Weg03] Ingo Wegener. "Methods for the analysis of evolutionary algorithms on pseudo-Boolean functions". In: *Evolutionary optimization*. Springer, 2003, pp. 349–369.

[WPN16] Junhua Wu, Sergey Polyakovskiy, and Frank Neumann. "On the Impact of the Renting Rate for the Unconstrained Nonlinear Knapsack Problem". In: *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016*. Ed. by Tobias Friedrich, Frank Neumann, and Andrew M. Sutton. ACM, 2016, pp. 413–419.

[Xu+07] Ke Xu et al. "Random constraint satisfaction: Easy generation of hard (satisfiable) instances". In: *Artif. Intell.* 171.8-9 (2007), pp. 514–534.

[Yan15] Shengxiang Yang. "Evolutionary Computation for Dynamic Optimization Problems". In: *Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015, Companion Material Proceedings*. Ed. by Sara Silva and Anna Isabel Esparcia-Alcázar. ACM, 2015, pp. 629–649.

[ZG99] G Zhou and M Gen. "Genetic Algorithm Approach on Multi-Criteria Minimum Spanning Tree Problem". In: *European Journal of Operational Research* 114 (1999), pp. 141–152.

[ZH07] Yuren Zhou and Jun He. "A Runtime Analysis of Evolutionary Algorithms for Constrained Optimization Problems". In: *IEEE Trans. Evolutionary Computation* 11.5 (2007), pp. 608–619.

[ZLT01] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. "SPEA2: Improving the strength Pareto evolutionary algorithm". In: *TIK-report* 103 (2001).

[ZT98] Eckart Zitzler and Lothar Thiele. "An evolutionary algorithm for multiobjective optimization: The strength pareto approach". In: *TIK-report* 43 (1998).

[ZV16] Haifeng Zhang and Yevgeniy Vorobeychik. "Submodular Optimization with Routing Constraints". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI 2016*. AAAI Press, 2016, pp. 819–826.