THE UNIVERSITY of ADELAIDE

Application of Bio-inspired Algorithms to Selected Real-World Problems

AUTHOR: Hirad Assimi

A thesis submitted for the degree of
DOCTOR OF PHILOSOPHY
The University of Adelaide

in the

Optimisation and Logistics
School of Computer Science

January 2023

Contents

List of Figures

List of Tables

Acronyms

ACO ant colony optimisation.

CCKP chance-constrained knapsack problem.

CCP chance-constrained programming.

DOP dynamic optimisation problem.

EA evolutionary algorithm.

EC evolutionary computation.

GA greedy algorithm.

ILS iterated local search.

KP knapsack problem.

MMAS max–min ant system.

MOEA multi-objective evolutionary algorithm.

MOOP multi-objective problem.

PSO particle swarm optimisation.

ROM run of mine.

SI swarm intelligence.

TSP travelling salesperson problem.

University of Adelaide

Abstract

Application of Bio-inspired Algorithms to Selected Real-World Problems

by Author: Hirad Assimi

Real-world combinatorial optimisation problems often include uncertain parameters, dynamic constraints, mixed solution representations, and precedence constraints that make finding high-quality feasible solutions difficult. Randomised methods such as bio-inspired algorithms can solve intractable combinatorial problems in a reasonable timeframe. Bio-inspired algorithms can solve a variety of problems by using operators inspired by nature. The thesis focuses on the practical application of bio-inspired methods to different combinatorial problems of varying attributes. We investigate four combinatorial problems, starting with the knapsack problem with dynamic inequality capacity constraint, which changes over time and random weights of items from a probability distribution. We use two inequality tail bounds, namely Chebyshev's inequality and Chernoff bound to derive helper objective functions to quantify the uncertainty for a candidate solution to the knapsack problem. We address the problem as a bi-objective problem to handle uncertainties, and we modify a baseline multi-objective optimisation algorithm to tackle the dynamic constraint to maintain separate partitions of infeasible and feasible solutions at each time frame ready to adapt to the capacity constraint change. Our experimental results show that adding a second objective to this complex problem results in better solutions than using a single-objective approach, and further improvements can be achieved using more sophisticated approaches such as NSGA-II. The second problem is a famous structural engineering problem whose ultimate solution as a structural design combines combinatorial and continuous components. We focus on the combinatorial aspects of the problem at the upper level of a bi-level problem setting, whereas at the lower level, we use a state-of-the-art optimiser to determine optimal components. We develop an enumeration method and a novelty-based approach to deal with small and large-scale problems. Using the proposed methods, we are able to obtain high-quality designs with similar objective functions and different features. The last two problems are concerned with solving a stockpile management problem in the mining industry, where a low-quality solution can result in the mining operator paying huge financial penalties to the client. Based on the data provided by our industry partner, we model this issue as a permutation problem. We use greedy algorithms and ant colony optimization to solve and understand it. Using a lexicographic objective function, we demonstrate how effectively these algorithms prioritise penalty fees over speedy delivery. We expand the previous problem to consider more realistic settings to schedule multiple machines, consider their safety constraints, and take material from stockpiles in a cooperative manner and demonstrate their efficiency.

Declaration of Authorship

I 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.

Hirad Assimi

August 2022

Acknowledgements

My deepest gratitude goes out to my supervisors, Prof. Frank Neumann and A/Prof. Markus Wagner for their invaluable guidance and unconditional support throughout my PhD journey.

Thank you to Mr. Ben Koch, Mr. Chris Garcia, and Mr. Paul Elson, our industry partners at Eka Software Solutions, for contributing and sharing their knowledge and technical expertise to help me understand mining operations.

Also, I would like to thank all the co-authors on my PhD papers, Dr. Aneta Neumann, Dr. Yue Xie, Mr. Oscar Harper, and Prof. Xiaodong Li, with whom I learned a lot.

I am grateful for the Research Consortia Program (RCP) "Unlocking Complex Resources through Lean Processing" for funding my studies and for providing a unique environment to grow among like-minded people. Thanks to my wonderful colleagues at the School of Computer Science and the Optimisation and Logistics group, I am very grateful for their support.

Last but not least, I would like to extend my sincere gratitude to my family and friends. Sandy, my life partner, deserves special thanks. Thanks for being patient with me and supporting me through the vicissitudes of this journey.

To my beloved family; Sandy, Sharareh, Hossein, Shamin and Seraj

Chapter 1

Introduction

Evolutionary computation (EC) is the area of computer science that uses bio-inspired search methods that mimic the evolution of biological entities or simulate the behaviour of natural biological evolution. There are various bio-inspired methods such as evolutionary algorithm (EA) [ES15], ant colony optimisation (ACO) [DS04] and particle swarm optimisation (PSO) [KE95].

Bio-inspired algorithms can be considered general problem solvers with a randomised population that model some natural phenomena. EAs use Darwinian evolutionary theory as survival of the fittest, ACO mimics the foraging behaviour of real ants, while PSO represents the collective learning of individuals or birds when they are in groups. Creating a population of individuals representing solutions to the optimisation problem is typically the starting point of these algorithms. Following that, the algorithms determine how well these individuals in the population solve or come close to solving the problem. Next, they use bio-inspired operators, such as mutation and selection, to create a new population using the current individuals as parents. The individuals in the next population inherit some characteristics of better-suited individuals and change randomly in an attempt to become fitter than their parents. The process continues until the termination criteria are met.

Finding the best solution among all feasible solutions is an optimisation problem. Combinatorial optimisation problems entail selecting the best solution from a finite set of potential solutions that satisfy particular properties. In practice, many combinatorial optimisation problems are NP-hard to solve optimally, and bio-inspired methods can explore these complex search spaces in a reasonable time while not proving global optimality. As a result, these algorithms can be used to obtain nearly optimal solutions to optimisation problems, which may fail to be solved using traditional mathematical techniques. Bio-inspired algorithms have been applied to a wide range of combinatorial and real-world problems, demonstrating the high capability in solving hard and complex problems [MA94; CWM12; NW10].

1.1 Contributions And Background

The primary motivation for this thesis is to conduct a practical investigation of EC for various real-world combinatorial problems with varying attributes. These attributes can refer to different types of constraints, objective functions, solution representation and the problem setting environment.

Many real-world combinatorial optimisation problems contain both stochastic and dynamic components. In an optimisation problem, the objective function, constraints, or decision variables may change as a result of dynamic components. EAs are the natural way to deal with dynamic optimisation problems (DOPs) because they are inspired by nature which is an ever-changing environment [NYB12]. Moreover, EAs can tackle the main difficulty in solving DOPs, which is tracking the moving optima

when changes occur. The Knapsack problem (KP) is a well-known combinatorial problem that can be defined as follows. Considering a set of items, each with a weight and value, how can we determine which items to include in a collection so that the total weight is within the given capacity and the profit is maximised?

In combinatorial problems, the behaviour of EAs has been studied using KP with dynamically changing constraint [RNN18]. Based on the trade-off between total profit and dynamic capacity constraint, they proposed a bi-objective optimization algorithm to track moving optima by maintaining feasible and infeasible individuals in the population within different partitions. They showed that dealing with the inequality constraint using a bi-objective approach can be advantageous in obtaining better solutions when the capacity change occurs less frequently. As an expansion of this work, they examined the EA behaviour of a range of problems based on their submodularity ratio [Roo+19].

Solving constrained optimisation problems subject to uncertainties is another attribute of optimisation problems. Failure to consider uncertainties could result in suboptimal or infeasible solutions in practice [LL15]. It is possible to deal with deterministic inequality constraints subject to uncertainty by turning them into chance constraints so that the probability of constraint violation is smaller than a predefined limit [CC59]. Chance-constrained programming (CCP) has been used successfully in various domains, including process control, scheduling, and supply management, where safety requirements are concerned [FGS16].

The weights in KP can be non-deterministic and be a random variable with regard to a probability distribution. Recently, Xie et al. [Xie+19] integrated inequality tails with single and bi-objective EAs to solve the chance-constrained knapsack problem (CCKP). The CCKP is a stochastic variant of the knapsack problem, in which weights are not known precisely but are randomly chosen from a distribution. We explain this problem in detail in Section 2.3. Chance constraints convert the deterministic constraint to a probabilistic one. They employed probability tail inequalities to estimate the likelihood of constraint violation. They also carried out bi-objective optimisation with respect to the profit and probability of constraint violation, considering the knapsack capacity is static. Doerr et al. [Doe+20] have investigated adaptations of classical greedy algorithms for optimising submodular functions with chance constraints of the knapsack type. They have shown that the adapted greedy algorithms can maintain asymptotically almost the same approximation quality as in the deterministic setting when considering uniform distributions with the same dispersion for the knapsack weights.

In Chapter 4, we consider the CCKP with dynamic capacity, where the weight of each item is chosen from a probability distribution and the capacity constraint changes dynamically over time. The goal is to maximise total profit subject to the probability that total weight does not exceed the capacity at a time step. We use well-known tail inequalities such as Chebyshev's inequality and the Chernoff bound to approximate the probabilistic constraint. Our key contribution is to introduce an additional objective that estimates the minimal capacity bound for a given stochastic solution that still meets the chance constraint. This objective helps to cater for dynamic changes to the stochastic problem. We apply single- and bi-objective evolutionary algorithms to the problem and show how bi-objective optimisation can help to deal with dynamic chance-constrained problems.

A solution to an optimisation problem can have both combinatorial and continuous decision variables. So optimisation problems may require explicitly modelling the interaction among different aspects of the problem. Truss optimisation problems are

well-known engineering problems in that the solution representation combines combinatorial with integer and continuous variables. Trusses are used in the construction of bridges, towers [Has07; Rao95], aerospace structures [Seb+11] or in robots [Fin+13]. Their primary function is to support structures under external loads. In truss optimisation, we aim to reduce the weight of the truss by considering topology, size and shape optimisation, where each has its own type of design variables.

Truss optimisation can be categorised into topology, size and shape optimisation. An objective of topology optimization is to determine what components should be included or excluded in the design of a truss, while size and shape optimization aims to find the best weight for each component. We aim to decide whether to include or exclude necessary truss members so that the structure's weight is as light as possible while adhering to structural constraints.

These structural constraints include stability, failure criteria and design codes by practice and manufacturing specifications. These constraints can conflict with objectives which makes the problem more challenging. For example, increasing the stiffness of the truss to prevent failure will increase its weight. Several numerical methods have been applied to different truss optimisation problems for decades. Conventional methods such as gradient-based showed limited efficiency in dealing with structural constraints and handling the discreteness and discontinuities in truss optimisation problems which led to trapping in local optima [Deb01]. The inadequacy of classical optimisation methods led to the development of EC techniques and metaheuristics in truss optimisation [KAD05].

Islam et al. [ILM17] adopted a bilevel representation for the truss optimisation problem to consider the interaction among different aspects of the problem. The bi-level formulation consists of an upper-level optimisation problem that determines the truss topology and a lower-level optimisation problem for determining the truss configuration. In both the upper and lower levels, the main optimisation objective was to reduce the weight of the truss. They used a modified binary PSO with a niching capability to increase population diversity while maintaining some level of exploitation in the upper level. It is based on a speciation scheme that uses a niching radius to subdivide the PSO population in order to locate multiple high-quality solutions. In order to supply the sizing solutions to the upper level, a standard continuous PSO was used at the lower level.

In Chapter 5 we introduce the exact enumeration to rigorously analyse the topology search space and remove randomness for small truss optimisation problems. We also propose novelty-driven binary PSO for bigger problems to discover new designs at the upper level by maximising novelty. For the lower level, we employ a reliable evolutionary optimiser to tackle the layout configuration aspect of the problem. The truss optimisation problem considers the manufacturability aspect where the sizing of bars should be chosen from a discrete set with respect to practice code constraints. We show that the proposed method outperforms the current state-of-the-art and can obtain multiple high-quality solutions for each instance.

Scheduling problems can be modelled as combinatorial problems. Real-world scheduling optimisation problems are subject to technical constraints that violation of these can severely damage the expectation of profit. Mining industry scheduling problems are challenging to solve in real-world supply chain management applications It involves a variety of technical constraints, and any delay in operations results in a loss of income. Mining is the process of digging rocks, extracting valuable material (ore), manufacturing it and finally selling it to obtain money [Sam+17]. EC techniques have been applied to challenging combinatorial and continuous optimisation problems in real-world applications of mining. EAs, for example, have been employed

in the iron mine supply chain optimisation for long-term planning of mine digger equipment and trucks for commercial purposes [Ibr+14]. ACO has been applied to solve the open-pit mining scheduling problem [SS15]. Recently, the differential evolution [SP97] has been used to tackle the stockpile blending problem considering the average grade of stockpiles and addressing a continuous optimisation problem subject to uncertainty [XNN21a; XNN21b]

Run-of-mine (ROM) stockpiles are essential components in the mining value chain, as they are used to temporarily store extracted ore before they are fed into the next stage. ROM stockpiles can balance inflows and outflows and assist in blending material for high-quality delivery. Stockpile schedulers plan stockpile recovery to balance throughput and material specifications to deliver for the supply chain's next stage [JHE13; Li+19]. Failure to meet technical parameters on delivery can result in significant penalties, increased operational costs due to poor operational plans or over-delivery of material specifications. Currently, human experts determine the planning of stockpile recovery in practice. However, this approach is error-prone due to the complex distribution of materials within a stockpile and the inability to foresee upcoming deliveries efficiently. Human decision-making can also result in a loss of assumed profits and disruptions in stockpile management.

In Chapter 6 we model the stockpile recovery problem as a combinatorial optimisation problem considering technical restrictions in real-world operations. This chapter investigates multiple scenarios and experiments with a single reclaimer in operation. We apply deterministic and randomised greedy algorithms as baseline algorithms and employ ACO algorithms integrated with local search to solve the problem. We compare all algorithms with a rule-of-thumb heuristic proposed by the industry partner to evaluate our methodology's quality.

We extend our investigation in Chapter 7 to include a more realistic scenario considering multiple reclaimers in action for short and long-term deliveries. The engagement of multiple reclaimers complicates the problem regarding their interaction in preparing a delivery simultaneously and safety distancing of reclaimers. We also consider other realistic settings, such as handling different minerals. We study various instances of the problem using greedy algorithms, ACO and propose an integrated local search method for determining an efficient schedule. We fine-tune the algorithm parameters and compare the algorithms in different settings to provide short to long-term schedules.

1.2 Outline Of The Thesis

This thesis mainly investigates four combinatorial optimisation problems: dynamic chance-constrained knapsack problem, topology optimisation of trusses and stockpile recovery optimisation problem using single and multiple reclaimers. These problems ranging from benchmark problems to real-world optimization problems can pose challenges due to uncertain parameters, dynamic constraints, mixed solution representations, and precedence constraints, making it difficult to find high-quality feasible solutions.

Chapter 2 describes these problems, and their characteristics in detail, including prerequisite knowledge on chance-constrained problems, structural optimisation and the real-world scheduling problem in the mining industry. Next, in Chapter 3 we introduce bio-inspired methods and other randomised and deterministic approaches used in this thesis. We explain several exact methods, including dynamic programming, iterated local search, and greedy deterministic search. Then, we describe the

EAs employed in this thesis, both single and bi-objective, before concluding with a description of PSO and ACO.

We start in Chapter 4 with the knapsack problem with dynamic inequality capacity constraint, which changes over time and random weights of items from a probability distribution. The main challenge with this problem is to track the optimal solution subject to dynamic constraints and quantify the probability of violating the knapsack capacity constraint. This chapter focuses on the investigation of EAs for the dynamic chance-constrained knapsack problem, where we introduce a new objective function to transform a single objective optimisation problem into a bi-objective problem. We derive helper objectives from the inequality tails to have two objectives. We solve the CCKP in dynamic settings with respect to the total profit and probability of violating the probabilistic constraint. We apply baseline single and bi-objective EAs to the problem. The analysis has been extended to include different complexities based on the amount of uncertainty, the dynamic environment, and the number of items in the knapsack instances.

Chapter 5 considers bilevel optimisation of trusses for topology, size and shape optimisation. The challenge with this problem is having a mixed solution representative for an ultimate design combining combinatorial and continuous components. Having mixed solution representatives can expand the search space resulting in finding a low-quality solution. In addition, considering mixed representatives avoids modelling the interaction among different aspects and makes it harder to evolve a solution efficiently. We employ exact enumeration for small-scale problems to analyse the search space rigorously. Next, we develop a novelty-driven binary PSO to explore the upper level search space for a more complex problem.

Chapters 6 and 7 focus on a real-world scheduling problem in the mining industry. In Chapter 6, we initiate the study by modelling the stockpile recovery problem as a combinatorial optimisation problem subject to technical restrictions in practice. The main challenge for this problem is having precedence constraints restricting the search space to find high-quality feasible solutions. We introduce a lexicographic objective function for the problem to minimise the penalty fees the producer will have to pay if the problem constraint is violated. We use optimisation methods to construct a schedule plan step by step to meet the precedence constraints in solving the problem. For this purpose, we explore using deterministic and randomised greedy algorithms. We also employ a variant of the ACO algorithm and consider three local search operators for it: swap, insert, and inversion for neighbourhood search of obtained solutions by ACO.

Next, in Chapter 7, we extend the stockpile recovery problem to consider multiple stockpiles and reclaimers and their interactions to provide deliveries. We schedule short and long-term deliveries while avoiding reclaimers crossing each other subject to more realistic settings than in the previous chapter. To simulate the interaction of the reclaimers in constructing a feasible solution, we develop a solution construction heuristic engaging multiple reclaimers. We investigate deterministic and randomised greedy algorithms and ACO to build the schedule step by step. We also use the automatic parameter tuning method to fine-tune the parameters of our algorithms to achieve better results. We also designed a local search operator for ACO to more finely investigate a solution, and it can outperform other methods in most instances.

Finally, Chapter 8 concludes the thesis with a summary of major findings.

Chapter 2

Combinatorial and Stockpile Recovery Problems

2.1 Introduction

Combinatorial optimisation is a branch of applied mathematics that deals with solving discrete optimisation problems. Combinatorial optimisation problems arise in various applications, such as operations research and artificial intelligence, which involve computational methods. These problems may include planning, scheduling, time-tabling, and resource allocation. Well-known objectives are to find the shortest or cheapest round trip in graphs, the optimal grouping, ordering, or assigning discrete and finite components.

The combinatorial optimisation problem is defined as a triple (S, f, Ω), where S is a search space, f is an objective function, and Ω is a set of constraints [NW10]. In a discrete search space, the main objective is finding solutions that meet all constraints and maximise or minimise the objective function. Candidate solutions are the result of attempts to solve a problem that is formed by combining solution components, but they may not satisfy all the conditions of the problem definition when compared to feasible solutions.

The computational complexity of a combinatorial problem instance is determined by the time and space it takes to solve it as a function of its size. Problem instance size refers to the length of a reasonably concise description [HS04]. Note that computation theory mainly deals with a set of problem instances as problem classes. It can be challenging to solve combinatorial problems because there are often many possible choices in the discrete search space. As the size of the problem instance increases, the number of candidate solutions will increase exponentially.

The $\mathcal{P}$ versus $\mathcal{NP}$ problem is a question in computer science that asks whether every problem for which it is easy to check a proposed solution can also be solved quickly (that is, in polynomial time). $\mathcal{P}$ is the set of problems that a deterministic Turing machine can solve in polynomial time. $\mathcal{NP}$ is the set of problems that can be solved by a non-deterministic Turing machine in polynomial time. It is widely assumed that $\mathcal{P}$ is not equal to $\mathcal{NP}$. The class of $\mathcal{NP}$-complete problems contains the hardest problems in $\mathcal{NP}$, and a problem is called $\mathcal{NP}$-hard if it is at least as hard as any problem in $\mathcal{NP}$.

This chapter begins with a description of the knapsack problem (KP) and presents a stochastic weighted variant of this problem in which the probability that the knapsack capacity is exceeded is bounded. We then present a combinatorial optimisation problem for topology optimisation of trusses with the aim of finding a subset of possible connections connecting truss members of a structure while reducing its mass. We then describe stockpile recovery as a combinatorial scheduling problem and explain its significance as a real-world combinatorial optimisation problem.

TABLE 2.1. Corresponding weight and profit interval for knapsack problems instances

type	weight (w_i)	profit
Uncorrelated	[1,1000]	[1,1000]
Bounded strongly correlated	[1,1000]	$w_i + ç$

2.2 Knapsack Problem

The knapsack problem is a classical NP-hard combinatorial optimisation problem. KP is defined as given n items where each item i, $1 \leq i \leq n$ has a profit p_i and a weight w_i and a knapsack capacity C. The objective is to find a subset of items that maximises the total profit ($P(x)$) while keeping the weight ($W(x)$) of the selected items within the knapsack capacity. The candidate solution is an element $x \in \{0,1\}^n$ where the item i is chosen iff $x_i = 1$. Therefore, KP is defined as

$$\begin{aligned} &\text{Maximise} && P(x) = \sum_{i=1}^{n} p_i x_i \\ &s.t. && W(x) = \sum_{i=1}^{n} w_i x_i \leq C \end{aligned} \tag{2.1}$$

There are many real-world applications of the knapsack problem, for instance when resources are limited and gaining a high reward is essential. Some examples include mining planning scheduling, optimisation of transportation systems, and network design [KPP04].

The correlation between profit and weight of items can result in different problem instances, namely weak and strongly correlated problems instances for the knapsack problem. The weight of items is selected randomly in *weakly correlated instances*, but the profit is chosen from a range in the vicinity of the selected weight. However, in *strongly correlated instances*, weights are random, but the profit is strongly correlated and determined by shifting the weight value. Table 2.1 shows the corresponding weight and profit for each type of knapsack instance where ç denotes a constant number. For more information on the generation of KP instances, see [Pol+14].

2.3 Chance-Constrained Knapsack Problem

The concept of uncertainty can be defined as the difference between the present knowledge and the state of complete knowledge [NGS05]. Generally, scientists and engineers classify uncertainties into aleatory and epistemic types. Aleatory uncertainty, also known as objective or stochastic uncertainty, results from the inherent randomness of a physical system or environment. In comparison, the source of epistemic uncertainty, subjective uncertainty, is the lack of knowledge about a physical system or environment. Probabilistic methods such as robust and reliability-based design are usually used to quantify aleatory uncertainty. On the other hand, non-probabilistic methods such as interval analysis and evidence theory are used to deal with epistemic uncertainty.

Aleatory uncertainty is a common factor in engineering design problems. Probability theory is used widely in engineering as a tool to manage uncertainty. However, the application of different methods relies on the complexity of the model of the problem. Different probabilistic techniques deal with this kind of uncertainty, including sampling and approximation methods [Li+16a].

Uncertainty can be taken into account when solving combinatorial problems. With the knapsack problem, uncertainties can be introduced into the weight of items due to incomplete information, the profit of items due to market fluctuations, or the knapsack capacity due to queuing problems [Kos14]. Considering the deterministic variant of KP, finding an optimal solution is certainly NP-hard.

Chance-constraint programming (CCP) is a powerful probabilistic tool to model uncertainty in optimisation problems. It transforms an inequality constraint into a probabilistic constraint to ensure that the probability of constraint violation is smaller than a limit pre-defined by the decision-maker [CC59]. By considering this feasibility threshold, an inequality turns into a chance constraint. CCP has been applied successfully in different domains such as process control, scheduling and supply management where safety requirements are concerned [FGS16].

A chance-constraint formulates an optimisation problem with random design variables integrating uncertainty while it satisfies a given threshold which is known as the reliability level [PAS09; Liu+13]. This type of constraint guarantees that the constraint is satisfied probabilistically with only a small fraction of violation [BS07; KN08; GR10]. Chance-constrained knapsack problem (CCKP) is a variant of the classical NP-hard deterministic knapsack problem where the weights and profits can be stochastic [KPP04].

In CCKP, the goal is to maximise the total profit of selected items $P(x)$ under the condition that the probability that the total weight of the selected items is at least as high as the capacity is at most α. Therefore, each potential solution for the problem should satisfy the probabilistic constraint where the probability of exceeding the knapsack capacity with random weights of items should be less than a pre-defined possibility of failure. It means that if we consider a high-reliability level in the constraint, such as 99%, there is only a 1% chance of failure. It means that the constraint only allows for 1% of failure, which is probabilistically determined.

Formally, we define the chance constraint as

$$\Pr\left[W(x) \geq C\right] \leq \alpha$$

where α is a parameter that upper bounds the probability of exceeding the knapsack capacity ($0 < \alpha < 1$). Therefore, for the chance-constraint knapsack problem, we have

$$\begin{array}{ll} \text{Find} & x \in \{0,1\}^n \\ \text{Maximise} & P(x) = \sum_{i=1}^{n} p_i x_i \\ \text{Subject to} & \Pr\left[W(x) \geq C\right] \leq \alpha \end{array}$$

Chance-constrained problems are hard to solve for multiple reasons. First, estimating the probability of constraint violation is usually difficult. Second, the feasible region due to this probabilistic constraint is usually non-convex [PAS09]. Chance-constrained optimization involves modelling uncertainty in the problem constraints. However, it can be difficult to accurately model the probability distribution of the uncertain data, which can make it challenging to estimate the likelihood of a constraint violation. Many chance-constrained optimization problems are non-convex, meaning that they have multiple local optimal solutions rather than a single global optimal solution. This can make it difficult to determine the optimal solution and estimate the probability of constraint violation.

Sampling techniques such as Monte Carlo Simulation produce random samples

considering a pre-defined probabilistic distribution for uncertain parameters. Simulation of the system considering these randomly generated samples is followed by minimising the standard deviation of the objective functions to achieve robustness. Sampling methods rely on sampling and are highly sensitive to the parameters or settings [Liu+13]. Furthermore, Monte Carlo simulations are computationally expensive [Lob+19]. Klopfenstein and Nace investigated the chance-constrained knapsack problem and provided an algorithm for stochastic knapsack problems when the profits are identical, or the uncertainty in the weight of items shares the same characteristic [KN08]. Goyal and Ravi proposed a polynomial-time approximation scheme when the weight of items is random variables normally distributed [GR10]. Han et al. proposed a pseudo-polynomial time algorithm to solve the chance-constrained knapsack problem. However, their approach may lead to infeasible solutions [Jin+16]. Another approach to address uncertainty in the knapsack problem is the relaxation analysis of stochastic knapsacks. This approach checks the bound of the knapsack after each item is assigned to determine how much capacity remains [BT19].

Recent research has focused on the use of evolutionary algorithms to solve chance-constraint combinatorial optimisation problems. Xie et al. [Xie+19] integrated inequality tails with evolutionary algorithms to estimate the constraint's violation probability. They reported on the behaviour of Chebyshev's inequality and the Chernoff bound for chance-constraint approximation in CCKP. They also carried out bi-objective optimisation with respect to the profit and chance-constraint while the capacity is static. Doerr et al. [Doe+20] have investigated adaptations of classical greedy algorithms to optimise submodular functions with chance constraints of the type of knapsack problem. They have shown that the adapted greedy algorithms maintain almost the same approximation quality as in the deterministic setting when considering uniform distributions with the same dispersion for the knapsack weights. Neumann and Sutton [NS19] analysed different settings with respect to the run-time analysis of CCKP using baseline algorithms. Xie et al. [XNN20] expanded their investigation by considering problem-specific crossover operators and heavy-tailed mutation operators. Monotone chance constraints optimisation using multi-objective optimisation algorithms has been investigated [NN20], and further research has been done on the run-time analysis of evolutionary algorithms applied to chance-constraint problems [Xie+21; NW21]. Moreover, EAs have been used to deal with a chance-constrained problem in a continuous optimisation problem in the mining industry [XNN21a]. Heuristic techniques have been developed to deal with uncertainty in the quality of extracted material from the mine and to ensure that the constraints of the resource and the client are met in planning.

2.3.1 Inequality Tail Bounds For Chance-Constraints

Quantification of probabilistic constraints can be difficult because it involves accurately estimating the likelihood of certain events occurring, which can be challenging due to the inherent uncertainty and variability involved in many real-world situations.

In a chance-constrained optimization problem, the probabilities of certain events occurring are used to determine the likelihood that the constraints of the problem will be satisfied. The type of probability distribution used to model these probabilities can affect the difficulty of the problem and the feasibility of the solution.

Chebyshev's inequality and the Chernoff bound are two of the most used inequality tail bounds. Chebyshev's inequality tail can determine a bound for a cumulative distribution function of a random design variable. Chebyshev's inequality requires knowing the standard deviation of design variables and gives a tighter bound than

weaker tails such as Markov's inequality. Therefore, it can be applied to any distribution if the expected weight and standard deviation are determined.

The standard Chebyshev inequality is two-sided and provides tails for the upper and lower bounds [CB02]. Cantelli's inequality, a one-sided Chebyshev inequality, is used in this thesis because we would like to investigate the upper-bound violation of the inequality constraint similar to the KP inequality constraint. For brevity, we refer to the one-sided Chebyshev inequality as Chebyshev's inequality in this thesis.

Theorem 1 (One-sided Chebyshev inequality)**.** *Let X be an independent random variable, and let $\mathbb{E}(X)$ denote the expected weight of X. Further, let σ_X^2 be the variance of X. Then for any $\lambda \in \mathbb{R}^+$, we have*

$$Pr[(X - \mathbb{E}(X)) \geq \lambda] \leq \frac{\sigma_X^2}{\sigma_X^2 + \lambda^2}.$$

Compared to Chebyshev's inequality, Chernoff bound provides a sharper tail with an exponential decay behaviour. In order to use Chernoff bound, the random variable must be a summation of independent random variables. Chernoff bound seeks a positive real number t in order to find the probability where the sum of independent random variables exceeds a particular threshold [MR95]. Therefore, Chernoff bound for an independent variable X can be given as follows based on Theorem 2.3 in [McD98].

Theorem 2. *Let $X = \sum_{i=1}^{n} X_i$ be the sum of independent random variables $X_i \in [0, 1]$ chosen uniformly at random, and let $\mathbb{E}(X)$ be the expected weight of X. For any $t > 0$, we have*

$$Pr[X \geq (1+t)\mathbb{E}(X)] \leq \exp\left(-\frac{t^2}{2 + \frac{2}{3}t}\mathbb{E}(X)\right).$$

$$\Pr_{\text{Chebyshev}} \equiv \Pr\left[W(x) \geq C\right] \leq \frac{\delta^2 \sum_{i=1}^{n} x_i}{\delta^2 \sum_{i=1}^{n} x_i + 3(C - \mathbb{E}(W(x)))^2}, \tag{2.2}$$

$$\begin{aligned} &\Pr_{\text{Chernoff}} \equiv \Pr[W(x) \geq C] \\ &\leq \exp\left(-\frac{3(C - \mathbb{E}(W(x)))^2}{4\delta\left(3\delta \sum_{i=1}^{n} x_i + C - \mathbb{E}(W(x))\right)}\right). \end{aligned} \tag{2.3}$$

2.4 Truss Optimisation Problem

Designing and developing structures to maximise profits from available resources has attracted much interest among scholars and has become a challenging and critical research topic over the last decades.

Trusses are structural frameworks consisting of truss members (bars) that carry only forces to their end nodes, and these bars can be made of steel, wood, or concrete. Trusses are commonly used to support roofs, bridges, and towers. Truss optimisation has been an active research area for decades. Due to its nature, it is a challenging problem and can provide a preliminary platform for evaluating methods for more complex design optimisation problems. The optimisation of trusses represents a complex and non-linear problem subject to constraints aimed at building a stable and practical truss. Truss optimisation problems are subject to multiple constraints such as stability, failure criteria, practice design codes, and manufacturing specifications. Conventional optimisation methods showed limited efficiency in solving the problem [DG01].

Truss optimisation problems can be classified into three categories: topology, size and shape optimisation. In topology optimisation, we aim to identify which truss

bars are redundant. In size optimisation, we search for the optimal size of truss bars used in trusses determined by their cross-sectional area, and in shape optimisation, we determine the optimum geometric coordinates of truss nodes in the design domain. Generally, the main objective function for truss optimisation is the weight of the structure, which is the sum of truss bar weights, where the weight of each included member depends on its length (determined by shape optimisation) and the cross-section area (determined by size optimisation). Minimising the mass of the truss is a preferred objective function for designers because the optimal design can reduce the design costs because of the lower material usage and easier transportation while simplifying the construction process.

Several numerical methods have been applied to different truss optimisation problems for decades. Conventional methods such as gradient-based showed limited efficiency in dealing with structural constraints and handling discreteness and discontinuities in truss optimisation problems, which led to trapping in local optima [DG01]. The inadequacy of classical optimisation methods led to the development of population-based algorithms and metaheuristics in truss optimisation [KAD05]. Various EAs have been used and developed for truss optimisation: Genetic algorithm [RK92; WC95; Zha+05], Differential evolution [Ho-+16], PSO [LHL09], Genetic Programming [Fen+14; Kha+20] and other metaheuristic algorithms [HE02; Che16; Lee+05; KT09; PB18; DLU18; DLU19; AD16].

There are two main approaches for truss optimisation in the literature with respect to EAs and metaheuristics: single-stage and two-stage designs. The two-stage methods assume that topology and size design variables are linearly separable, and initially find optimal topology considering fixed equal size for all active bars. Next, it determines the optimal sizing for the obtained topology. This naive assumption of being linearly separable could lead to missing out on good solutions and may result in infeasible solutions with respect to real practice, and this is a limitation of the two-stage method [DG01]. Single-stage approaches, on the other hand, take topology, size, and shape optimisation into account simultaneously, thus inevitably ignoring the interactions that may exist among different variables for different truss optimisation categories.

Therefore, modelling the interaction among different aspects of the truss optimisation problem is essential. Bilevel optimisation can tackle this main issue of the previous two approaches to deal with truss optimisation problems. In the bilevel formulation, the upper level optimisation problem determines the truss configuration, such as topology, where the lower level optimises bars' sizing and shape. Optimisation of the topology of trusses is a combinatorial and multi-modal problem that we consider to be the upper level of bilevel optimisation, while the lower level involves optimisation of the size and shape of the truss in the lower level.

Problem Statement

Truss optimisation problems are subject to structural constraints such as stability, failure criteria and design codes by practice and manufacturing specifications.

The ground structure method is a prominent approach to truss optimisation. It gives excessive potential connections between the nodes and the preliminary positions of the nodes in the design space [Top83]. Achieving an overall minimum weight of a truss is the most common objective in truss optimisation problems based on the structure of the ground. Light weight beams can save manufacturing resources and related costs, and in aerospace applications, lightweight structures can reduce the fuel burn rate [BKM17].

The constraints in truss optimisation can conflict with the objectives, making the problem more challenging. For example, lighter trusses can increase and increase the instability of the trusses and lead to failure. Trusses can also be subject to independent multiple external load cases in practice where the optimal design should perform well for all cases.

$$
\begin{aligned}
&\text{Find} && \mathbf{A}, \xi \\
&\text{Minimise} && W(A,\xi) = \textstyle\sum_{i=1}^{\hat{m}} \rho_i l_i A_i \\
&\text{subject to} && G_1 : \textit{Truss contains all the necessary nodes}, \\
& && G_2 : \textit{Truss is externally stable}, \\
& && G_3 : \sigma_i(\mathbf{A},\xi) \leq \sigma_i^{\max}, \quad \forall i \in \{1,2,..,\hat{m}\} \\
& && G_4 : \delta_k(\mathbf{A},\xi) \leq \delta_k^{\max}, \quad \forall k \in \{1,2,..,n\} \\
& && G_5 : A_{min} \leq A_i \leq A_{max}, \quad \forall i \in \{1,2,..,\hat{m}\} \\
& && G_6 : \xi_{min} \leq \xi_k \leq \xi_{max}, \quad \forall k \in \{1,2,..,n\}
\end{aligned}
$$

where $W(A,\xi)$ denotes the weight of the truss, $\mathbf{A}$ and ξ show the cross-sectional area of the truss bars and the nodal coordinates of nodes, respectively. $\hat{m}$ and n denote the set of bars and nodal coordinates available in the design space. ρ_i, l_i show the density of the material used for the ith bar and the length of the bar in the design space with respect to the nodal coordinates of the bar's joint. Note that if the problem excludes shape optimisation, then the length of bars is fixed due to no change in the position of nodes in the design space.

The ground structure of the problem determines the necessary nodes, which include the nodes that carry a load or are fixed to the ground (or rolling). Constraint G_1 enforces that all necessary nodes should be present in a feasible design. G_2 is to ensure kinematic stability of the truss, which checks the positive definiteness of the stiffness matrix of truss design. G_3 and G_4 ensure that the stress in bars (σ_i) and the displacement of nodes (δ_j) due to loading does not exceed the maximum allowable threshold as $\sigma^{\max}$ and $\delta^{\max}$, respectively. Constraints G_5 and G_6 ensure that the value of cross-section area and nodal coordinates remain within the allowable limits.

2.5 Multi-Objective Optimisation Problems

Many real-world applications require optimising multiple objective functions that we refer to as multi-objective optimisation problems (MOOPs). A general MOOP aims to find a vector of decision variables (x) to provide an optimal solution to all objective functions ($f_m(x)$). The following equation defines the general form of a MOOP.

$$
\begin{aligned}
&\min && f_m(x) && m = 1,2,\ldots,M \\
&\text{Subject to} && g_j(x) \leq 0 && j = 1,2,\ldots,J \\
&\text{Subject to} && h_k(x) = 0 && k = 1,2,\ldots,K
\end{aligned}
$$

The quality of a single-objective optimisation solution can be easily assessed by comparing its objective function values with those of other solutions. However, in MOOPs, one solution may be better for one objective but worse for another. A set of solutions is a typical answer to a MOOP where the set defines the best trade-offs between competing objectives.

We define *dominance* to determine the goodness of a solution as follows for a minimisation problem. Solution x dominates y ($x \succeq y$) if solution x is no worse than y in all objectives, and solution x is strictly better than y in at least one objective.

Therefore y is dominated by x. Given a set of solutions, the non-dominated set of solutions contains all the solutions that can not be dominated by any member in the non-dominated set.

Pareto-optimal set defines the set of non-dominated solutions of the entire feasible decision space, and mapping all solutions of the Pareto optimal set from the feasible decision space to the feasible objective space defines the boundary as *Pareto-optimal front*. So, MOOP aims to find diverse solutions as close as possible to the Pareto optimal front.

The weighted sum method is a classic approach to MOOP where the set of objectives is scalarised into a single objective by summing each objective multiplied by a pre-defined weight where the weight is chosen with respect to the relative importance of the objective. It is a simple method, but the main disadvantage lies in setting the weights. Another limitation is that the weighted sum method cannot find a real Pareto-optimal front for non-convex problems because the weighted sum method cannot find a non-dominated set of solutions and varying the weights does not necessarily result in reaching the representation of the Pareto front.

$$\begin{array}{lll} \text{min} & F(X) = \sum_{m=1}^{M} w_m f_m(x), & m = 1, 2, \ldots, M \\ \text{Subject to} & g_j(x) \leq 0 & j = 1, 2, \ldots, J \\ \text{Subject to} & h_k(x) = 0 & k = 1, 2, \ldots, K \end{array}$$

Another common method is the $\epsilon-$Constraint method, which maintains one of the objectives and converts the remaining objectives into constraints with an upper bound user-specific threshold.

This approach can be applied to the case of convex and non-convex problems, but the value for ϵ should be chosen carefully to be within the minimum or maximum ranges of the individual objective functions.

$$\begin{array}{lll} \text{min} & f_u(X) & \\ \text{Subject to} & f_m(x) \leq \epsilon_m & m = 1, 2, \ldots, M \wedge m \neq u \\ \text{Subject to} & g_j(x) \leq 0 & j = 1, 2, \ldots, J \\ \text{Subject to} & h_k(x) = 0 & k = 1, 2, \ldots, K \end{array}$$

In addition to solving problems with conflicting objectives, several studies have indicated that transforming a single-objective optimisation problem into a multi-objective optimisation problem may lead to obtaining better solutions. This transformation leads to obtaining a set of non-dominated solutions instead of a single solution. Therefore, each individual in the Pareto front can contain useful information that can improve the performance of the algorithm to explore the search space [NW06; QYZ15; Qia+17]

2.6 Stockpile Recovery Problem

The supply chain connects a manufacturer with the resources needed to produce a product. Human resources, infrastructures, and related data are some of these resources. In supply chain management, the optimal process steps should be considered in order to produce a product with a lower cost and higher quality. This can be achieved through scheduling optimisation.

In recent decades, optimal planning and scheduling have attracted numerous researchers. In order to maximise the benefits of resources, the manufacturer must develop a schedule that reduces costs and increases profits. Real-world scheduling

problems, such as those in the mining industry, are complex. An example of mining would be to dig blocks, extract valuable material (ore), manufacture it and then sell it for profit [Sam+17]. The huge haul trucks transport the ore from the mine development area to the Run-of-Mine (ROM) stockpiles for temporary storage, or the ore is sent directly to the crushers for processing. ROM stockyards are the place where minerals are deposited, and a stockyard consists of several stockpiles of different materials with different characteristics. ROM stockpiles are common in copper production and dry bulk terminals in iron and coal production.

Depending on the customer's requests, a selective addition of ore from stockyards should be selected at the required tonnage. In the stockyard, stackers and reclaimers are the machines that load and unload the ore in the stockyard, respectively. This selection process is a complex task subject to multiple operational constraints, such as when undesirable mineral properties should not exceed certain limits in a request. The stockpile schedulers select a blend of the material in ROM stockpiles to be sent to the crushers for processing. We refer to the selected minerals as *delivery* and the operation of taking material from the stockyard as *stockpile recovery*. Quality control of deliveries is a major concern, as higher-grade ore is frequently blended with lower-grade ore to guarantee that the delivery matches the customer's mineral needs. These deliveries should meet the required lower and upper bound target quality on the chemical components and meeting the required tonnage. Therefore, this selection is difficult due to the complexity of the ROM stockyard deposits, such as various irregularities in minerals.

Stockpile schedulers aim to figure out how to plan stockpile recovery to keep upcoming deliveries qualities consistent in terms of economic minerals, meeting clients' requirements and lowering operation costs. In practice, human specialists plan the stockyard recovery using the rules of thumb based on available data from laboratory samples and a rough estimate of the average grade of ore in the stockpile. However, this is an error-prone decision-making process with insufficient decision support due to the technical challenges that make it difficult to account for planning a long-term stockpile recovery. For example, a rule of thumb for recovery could be that the good quality ore should be mixed with some poor quality ore, as it helps meet the target concentrations as closely as possible. However, human planning is very challenging because there exist complexities in the stockyard quality models, such as multiple operational constraints to schedule how the stockpiles should be reclaimed. Human planning might lead to early exhausting of the available good material in the stockyard and in an unexpected loss. Furthermore, human planning can result in limited decision support and ability to consider the upcoming blends and requests.

Stockpile recovery is challenging due to technical restrictions, including chemical concentrations and operational constraints. Target quality specifications should be met in stockpile recovery. These specifications can be typical chemical concentrations and particle size distributions. Size limits avoid overwhelming crushing equipment with excessive material that is hard to crush. Preserving the chemical concentrations of the target quality is essential because the provider aims to deliver material consistently close to the target grade considering the contaminants that can lead to a penalty fee. Hence, these specifications must not exceed the limit bounds; or otherwise, it incurs a financial penalty for the provider. Another essential factor for the provider is the efficiency of performing reclamation operations to increase throughput and maximise profitability. Recovery operation by machines and their movement on the site can be expensive with respect to the operational cost. It is desirable for the stockpile schedulers to meet the target quality and reduce the operating costs to maximise profit.

Therefore, optimal planning of the reclaiming and stacking sequences in practice can be highly valuable in reducing variability in decision-making and operation costs. A poor stockpile recovery plan can have serious consequences. Substantial penalty fees, a lack of consistency in decision-making and operations, increasing operating costs, unanticipated losses in practice, and perturbations in the assumed profit can all be consequences. Stockpiles are critical components in mining supply chains since they potentially increase the net present value by maintaining a mining buffer.

Reclaimer scheduling in dry bulk terminals has been well explored [Ang+16; UO19], where they are mainly a variation of the parallel machine scheduling problem. However, the main shortcoming in these studies is that they adopt a supply chain model that considers the stockpiles as a whole. However, different areas in a stockpile can have varying mineral compositions in practice. Stockpile recovery optimisation considering cuts in the stockpiles has been studied in [LM11; LM10]. Their model considers the minimisation of the movement by a bucket wheel reclaimer machine. For this purpose, they calculated the movement by Euclidean distance and applied mixed integer programming for a small scale of the problem considering two requests and two stockpiles in a single period.

In our study, we show how previous studies are limited for a real-world application. To simulate the stockyard, we use the information provided by our industrial partner. They use a stockyard management software that tracks machinery using various data feeds such as GPS location as well as using a material flow modelling to create a high-resolution 3D model of the stockyard. Their software can estimate the mineral composition estimate for different parts of a stockpile which we refer to as *cuts*. Cuts are stacked portions of stockpiled material that represent different grades of mineral resources. Cuts vary in chemical composition and in tonnage. The software can simulate many reclaim operations in advance in order to provide the data and scenarios for optimisation.

Finding feasible scheduling solutions is also limited by precedence constraints. It is not possible to reclaim a lower level cut in a discretised stockyard before the corresponding cut above it has already been reclaimed. The dependency table captures the precedence constraints in the stockyard considering the cuts. These precedence constraints indicate that some cuts are impossible to be accessed before another cut as their precedence is reclaimed.

Stockyard recoveries include moving machines from one point to another in a stockyard and reclaiming material at the destination. Depending on the length of the travel and the size of the cut, the travel time will vary with respect to the duration of machine moving and reclaiming the destination cut, respectively. Our industry partner simulator calculates the recovery time.

All possible moves can be pre-calculated and assigned in a full square matrix, namely a time matrix where each element indicates the recovery time of an operation from one cut to another in the stockyard. It should be noted that if a reclaimer operation takes more time, it is a more expensive operation with respect to the operation costs.

There is currently a lack in the literature for dealing with stockpiles in realistic scenarios. When multiple reclaimers are present, the reclamation direction can shift, different types of material must be delivered, and the stockpile manager should deal with longer-term delivery plans. We investigate the stockpile recovery problem for single machine scheduling and multi-scheduling machine settings in Chapters 6 and 7, respectively. We consider this real-world problem as a combinatorial scheduling problem, and we introduce a lexicographic objective function where the primary objective for the stockyard manager is to avoid paying financial penalty fees with respect to the

quality objectives of deliveries. We aim to develop an effective schedule for reclaimers considering quality objectives and avoiding the penalty fees.

Precedence relationships between cuts limit the search space by restricting how cuts can be accessed regarding their position in the stockyard. To guarantee that the precedence restrictions are met in a final solution, we must ensure that every schedule segment is valid. Therefore we use solution construction heuristic approaches that can construct a solution step by step. We also need a method that allows us to deal with real-world problems that address different objective functions and constraints and effectively incorporate additional technological limitations based on end-user requirements for future additions.

2.7 Conclusions

In this chapter, we briefly introduced the combinatorial optimisation problems used in this thesis. We presented the formal definition of the chance-constrained knapsack problem where in Chapter 4, we add the dynamic component to this problem to deal with it in a holistic approach. The next step was to state the problem and review the literature on the truss optimisation problem. In Section 5, we develop exact and randomised methods to analyse the topology optimisation of trusses. We then explained a real-world issue related to mining scheduling in the mining industry. In Chapter 6, we investigate the stockpile recovery problem using a single reclaimer subject to basic assumptions and in Section 7, we extend our study by relaxing these assumptions and considering a more-realistic setting to schedule using multiple reclaimers.

Chapter 3

Deterministic and Randomised Methods for Optimisation Problems

3.1 Introduction

Algorithms for optimisation fall into two broad categories. They are deterministic and randomised algorithms. The deterministic algorithms always produce the same results given a particular input using the same computation steps. On the other hand, randomised algorithms make random choices while performing the computation steps to produce a result. So, given a particular input, the behaviour of algorithm can be different in multiple executions.

For simple problems, deterministic algorithms can find a solution quickly and correctly. However, difficult problems can take a long time to solve, or high-quality solutions may not be found in a reasonable time frame. Randomised algorithms can thus help solve problems by using random numbers. NP-hard problems are intractable optimisation problems. In other words, a deterministic Turing machine cannot solve any of the problems' worst cases in polynomial time [Sip97].

Despite their intractability, there are deterministic approaches like dynamic programming, constraint programming, and mathematical programming [Pin16]. For instance, we can solve the knapsack problem (see Section 2.2) using dynamic programming in pseudo-polynomial time, which suggests the knapsack problem consists of many sub-problems that contain duplicate calculations. The main limitations of deterministic approaches are the size and computational cost of the problems. Thus, cutting-plane methods are used in mathematical programming to reduce problem size. [DFJ54].

As an alternative, randomised search algorithms can tackle these limitations to solve NP-hard problems by employing randomness in their iterative search procedure. The global optimality of solutions is not guaranteed, but they can find a good solution in reasonable computation times for difficult problems. Bio-inspired algorithms belong to this group, where they use computing methods inspired by nature. Bio-inspired methods have been successfully applied in solving combinatorial optimisation problems [NW10; Pol+14; NW06].

In this chapter, we discuss the deterministic and randomised approaches used in this thesis. Dynamic programming is presented as an efficient deterministic approach to the knapsack problem. Iterated local search (ILS) is a tool to assist and improve the randomised algorithm search, particularly ACO. Following that, we present greedy algorithms, which help us understand combinatorial problems and construct a feasible

Algorithm 3.1: sub-problem procedure of dynamic programming procedure for binary knapsack problem

```
1 foreach subset K ⊆ I do
2     if the profit of K is the greatest so far and the weight of K is smaller or
       equal to C then
3         store K
4 return the best K
```

solution step by step. We then discuss randomised algorithms used in this thesis, which are single and bi-objective EAs, as well as ACO and PSO.

3.2 Deterministic Approaches

In this section, we describe the three exact approaches used in this thesis.

3.2.1 Dynamic Programming

Dynamic programming is suited to problems that can be formulated into several sub-problems; therefore, it can employ a divide-and-conquer approach to repeatedly solve the duplicate sub-problems. Recall the knapsack problem (KP) from Section 2.2, the basic exhaustive solution for KP is to try all combinations of the item set to choose the one with the highest profit subject to the knapsack bound. The algorithm must form a decision tree to try all the combinations at each step that it needs to decide whether to include or exclude an item to make a decision in $\mathcal{O}(2^n)$ where n refers to the number of items. This is an exponential function and intractable.

Polynomial-time algorithms are the class of algorithms whose order-of-magnitude time performance is bounded from above by a polynomial function of n, where n refers to the size of the input. While exponential-time algorithms are bounded from below and above by an exponential function of n. Polynomial algorithms are known as good algorithms as they provide an efficient surrogate and computation time and have been identified as a desirable feature for good algorithms [Sip92].

Turning KP problem into sub-problems is dependent on how many items are being considered and how much of the remaining weight can be stored; therefore, the storing table is two-dimensional. Considering an item set I with capacity C, we can define the division of the problem into sub-problems (see Algorithm 3.1). Therefore we can keep the intermediate results in a two-dimensional table P where each element of the table denoted by $P_{i,j}$ represents the highest knapsack profit for capacity j considering only the first i items.

The recursive step includes that we calculate the elements of the table in a bottom-up manner where w_i and p_i show the weight and profit of each item, respectively. Therefore for each item i, and capacity j, if adding the item does not exceed the capacity, we either include or exclude the item. If we exclude, we take the profit computed from the sub-problem excluding this item as $P_{i-1,j}$. Otherwise, if we include the item, we need to add p_i in addition to all the profits from the remaining items as $p_i + P_{i-1,j-w_i}$. Therefore the maximum profit is calculated by including or excluding this item. We calculate the maximum profit for the first i items with capacity j as

Algorithm 3.2: Dynamic programming for binary knapsack problem

```
foreach j = 0 to C do
    P_{0,j}=0
foreach i = 1 to n do
    foreach j = 0 to C do
        if j ≥ w_i then
            P_{i,j} = max(P_{i-1,j}, p_i + P_{i-1,j-w_i})
        else
            P_{i,j} = P_{i-1,j}
return P_{n,C}
```

follows in deciding the maximal gain that can be obtained.

$$P_{i,j} = \max \begin{cases} P_{i-1,j-w_i} + p_i & \text{item } i \text{ in the subset} \\ P_{i-1,j} & \text{otherwise.} \end{cases}$$

Algorithm 3.2 shows the dynamic programming procedure for the knapsack problem (defined in Section 2.2) The maximum profit could be exponential in size n, but this approach is a fully polynomial time approximation (FPTAS) for the binary knapsack problems [KPP04]. FPTAS is a type of algorithm that can approximate the solution to a problem within a given error margin and run time, which is polynomial in the size of the input. An example of an FPTAS is the Knapsack problem, which is an optimisation problem where the goal is to fill a knapsack with items of different weights and values in such a way that the total value is maximised, subject to a constraint on the total weight. An FPTAS for the Knapsack problem can be used to approximate the optimal solution to the problem for large inputs in polynomial time. Research has been conducted on finding faster FPTASs for binary knapsacks for a long-time and is summarised in [Jin19].

3.2.2 Iterated Local Search

Local search is a heuristic used to solve combinatorial problems. Local search is iterative and starts with a feasible initial solution (x). It considers the local neighbourhood $N(x)$ and attempts to find a better quality solution ($x*$) which is a perturbation of a previous solution. When the neighbourhood is small, there could be a rapid convergence to a local optimal solution, while when the neighbourhood is large, it is possible to choose a new solution that is very different from the current one. Therefore, the local search involves changing a solution locally to move it in the search space and evaluating different solutions until the optimal solution is found or the stopping criteria are met.

In this thesis, we employ the Iterated Local Search (ILS) [LMS03] which has been successfully applied to solve scheduling problems [DS16]. ILS can escape from a local minimum, and perturbations are a key part of an ILS framework because they provide the opportunity to jump to a new region in the search space. The distance between this new region and the current local optimum will depend on the type of perturbation involved. In this thesis, we consider an exhaustive ILS and we try all possible combinations according to the neighbourhood definition. We consider three well-known operators for permutation search problems namely *swap*, *insert* and

Algorithm 3.3: Iterated local search

```
1 x_0 = GenerateInitialSolution
2 x* = LocalSearch(x_0)
3 while StopCriteria ≠ True do
4     x' = Perturb(x*)
5     x*' = LocalSearch(x')
6     x* = Accept(x*')
7 return x*
```

inverse operators. Considering one solution for a combinatorial problem as x, these operators get two components in x and perform a local search on the components' position. The swap operator exchanges two components of the solution; the insert operator shifts the second component ahead of the first component. The inverse operator arranges all components between and including the two components in the opposite order.

Algorithm 3.3 shows the procedure for ILS, where *GenerateInitialSolution* refers to generating a feasible initial solution for ILS. The common way to obtain the initial solution is to use the output of a randomised algorithm, such as ACO. *LocalSearch*, *Perturb* and *Accept* are three key components of ILS algorithm. These procedures must complement each other in order to achieve a good trade-off between intensification and diversification [HS05]. *Perturb* is the process of making changes to the current candidate solution without immediately undoing them by subsequent local search phases. *Accept* determines between two solutions (x' and $x^{*\prime}$) which one is better to be accepted. Keeping the better solution leads to iterative improvement in the local search. However, if ILS chooses the perturbed solution regardless of the quality, it leads to a random walk in the local space. The *LocalSearch* determines the algorithm to be used for local search where it can be ILS itself or other local search algorithms.

3.2.3 Greedy Algorithms

Greedy algorithms (GA) have been used in solving many optimisation problems. They are fast and easy to interpret. For example, in the travelling salesperson problem (TSP), given a graph of vertices and weight function (according to the length of route), the objective is to construct a tour (a route that travels along each vertex exactly once) of minimum total weight. The most straightforward algorithm initiates with a vertex and selects one of its nearest neighbours (irrespective of the quality) as the next destination. The algorithm repeats this procedure till all vertices are visited. The more efficient greedy algorithm for TSP with respect to runtime is when the algorithm constructs a tour step by step by selecting the edge with the shortest length. The algorithm terminates when a Hamiltonian cycle has been found. The greedy algorithm for the travelling salesperson problem could potentially return bad solutions.

Using GAs, a complete solution can be constructed step by step. GA picks the successor component that is most likely to bring the greatest value at each time step when choosing which components to include in the solution. The procedure goes on for the next components until a full solution is generated. Deterministic greedy chooses the successor component with the highest greediness. As a result, at each time step, the component with the highest reward is selected. However, it has been shown that

Algorithm 3.4: Generic Deterministic Greedy Algorithm

```
1 X ← ∅
2 Evaluate the incremental cost of each component e ∈ ℰ
3 while X is not a complete feasible solution do
4 |   Choose a successor component x* ∈ ℰ with the highest profit
5 |   X ← X ∪ x*
6 |   Update the incremental cost of each component e ∈ ℰ \ X
7 return X
```

adding randomness into the selection procedure can result in finding a better solution [Gao+18]. Algorithm 3.4 shows the generic deterministic greedy algorithm procedure. In deterministic GA, the candidate set is initially empty. The next step is to evaluate the cost of adding each component that is available. It selects the component with the highest reward out of the candidates. It then updates the partial cost for the remaining components. This procedure repeats till the solution is completed.

3.3 Randomised Algorithms

This thesis defines randomised search algorithms as those that utilise either randomisation, probabilistic operators, or a combination of both. In this section, we introduce a couple of EC methods.

EAs are randomised search algorithms that use mutation, recombination and selection, which are the mechanisms of biological evolution. An individual in a population is the candidate solution to the optimisation problem. In EAs, individuals with high fitness values are considered to be potential parents of the next generation based on the objective function. Eventually, the population evolves after repeated application of the operators above.

Note that evolutionary computation is a subfield of artificial intelligence that focuses on the simulation of natural evolution to solve computational problems. It includes several nature-inspired algorithms such as genetic algorithms, evolutionary strategies, and genetic programming, among others.

Swarm Intelligence (SI) refers to the artificial intelligence techniques of social insects, animals, and human societies [Li+19]. The agents form an SI population, which communicates and coordinates with each other through simple interactions, such as exchanging information or reacting to the actions of their neighbours. Through these interactions, the agents can collectively solve problems and make decisions that are beyond the capabilities of any individual agent. Biological examples can include ants that demonstrate path-finding behaviours when they go out to find food and birds that fly in flocks to improve their survival chance when they travel. Swarm intelligence has been applied widely to continuous and combinatorial optimisation problems. In this thesis, we use two algorithms inspired by the phenomena observed in nature as PSO and ACO algorithms, respectively.

3.3.1 Evolutionary Algorithms

In evolutionary algorithms (EAs), survival of the fittest practices are used to solve a problem. EAs are inspired by biological operators such as reproduction, mutation, recombination and selection. The basic principle behind EA is as follows. Given a population of candidate solutions and an evaluation function that determines their

qualities, the natural selection driver of EAs leads to producing new offspring, and as a result, the fitness of the entire population is improved. EAs have been applied successfully to provide good solutions (approximation) in various domains such as engineering, science, arts, and economics.

A candidate solution to the optimisation problem is referred to as an individual in the population and is stored in a certain data structure. The qualitative evaluation measurement function is the *fitness function*. In each iteration, variation operators generate new individuals to form an offspring pool. The selection mechanism determines what individuals (out of the population and offspring pool) are preserved in the next iteration.

The general framework for EA methods can be defined as follows.

- Search Space: This is the set of all possible solutions to the optimization problem. Each point in the search space corresponds to a possible solution.
- Fitness Function: This is a function that assigns a value to each point in the search space, indicating how good that solution is. The fitness function is problem-specific and is used to evaluate the quality of solutions.
- Population: A population is a set of solutions, often represented as a set of points in the search space. These solutions are generated and updated over time by the algorithm to converge to an optimal solution.
- Initialisation: This step involves generating an initial population of solutions.
- Selection for reproduction: This step involves selecting a subset of solutions from the current population that will be used to generate the next population.
- Generation of Offspring: This step involves generating new solutions, called offspring, from the selected solutions. This can be done through various techniques such as crossover, mutation, or heuristics, depending on the algorithm.
- Evaluation: This step involves evaluating the quality of the new offspring solutions using the fitness function.
- Selection of Survivors: This step involves selecting a subset of solutions from the current population and the new offspring that will be used to form the next population. This can be done through various environmental selection methods such as elitist or truncation selection.
- Termination: This step is where the algorithm stops. The termination can be met by reaching a stopping condition such as reaching a maximum number of generations or reaching a solution that meets a certain fitness threshold.

In this section, we explain (1+1)-EA and GSEMO as baseline EAs used in combinatorial optimisation problems, including a well-known multi-objective EA known as NSGA-II.

(1+1)-EA

(1+1)-EA is a baseline single objective EA including only one individual in its population. This individual undergoes mutation in each iteration and the mutated solution will replace the current solution if it is determined to be at least as good or better according to the fitness function. Considering a binary representation of the solution, the mutation operator flips each part of the individual with a small probability with respect to the length of the individual. Algorithm 3.5 shows (1+1)-EA procedure.

Algorithm 3.5: (1+1)-EA

generate $x \in \{0,1\}^n$ uniformly at random
while *termination criterion not satisfied* **do**
  $y \leftarrow$ create an offspring by flipping each bit of x independently with the probability of $\frac{1}{n}$
  if $f_{(1+1)}(y) \succeq f_{(1+1)}(x)$ **then**
    $x \leftarrow y$
return x

Algorithm 3.6: G-SEMO

generate $x \in \{0,1\}^n$ uniformly at random
$S \leftarrow \{x\}$ **while** *termination criterion not satisfied* **do**
  $y \leftarrow$ create an offspring by flipping each bit of x independently with the probability of $\frac{1}{n}$
  if $\nexists w \in S : w \succeq y$ **then**
    $S \leftarrow (S \cup \{y\}) \setminus \{z \in S | y \succeq_{GSEMO} z\}$
return x

3.3.2 Multi-Objective Evolutionary Algorithms (MOEAs)

Multi-objective EAs (MOEA) are widely accepted and applied to real-world MOOPs. MOEAs deal with several (conflicting) objectives and provide a set of solutions [Deb01; QYZ15; Qia+17]. Similar to the single-objective EAs, MOEAs contain a population that evolve through generations using bio-inspired operators. But instead of finding a single solution, their solution is the Pareto set containing a set of non-dominated solutions. MOEAs can be elitist or non-elitist, where the latter does not use any elite-preserving operator. However, elitist operators carry the elites of the population to the next generation.

G-SEMO

The global simple evolutionary multi-objective algorithm (G-SEMO) [Gie03] is the multi-objective form of (1+1)-EA that similarly has been comprised of a single individual in the population and, it also opts for the same mutation operator as (1+1)-EA. However, it incorporates non-dominance and Pareto concepts into the algorithm to deal with multi-objective functions. Algorithm 3.6 shows its procedure.

NSGA-II

Non-dominated Sorting Genetic Algorithm II [Deb+02] is an elitist MOEA where the EA used in the algorithm uses crossover and mutation [Mit98].

NSGA-II iteratively puts the non-dominated solutions in the population to different fronts with equal domination [ES15]. It also uses a crowding distance metric defined for each individual where it is computed as the average side length of the cuboid with respect to the nearest neighbours in the front of the solution. NSGA-II choose parents for evolutionary operators using a modified tournament operator considering the front rank and crowding distance in order. NSGA-II operators enable

Algorithm 3.7: Non-dominated Sorting Genetic Algorithm II (NSGA-II)

$t \leftarrow 0$ ▷ `Generation counter`
while *StopCriteria* $\neq$ *True* **do**
  $P_t \leftarrow$ Initial_Population() ▷ `Generate random population with size` N
  $Q_t \leftarrow Offspring\_Population()$
  $R_t \leftarrow P_t \cup Q_t$ ▷ `Generate the offspring population with size` N
  Compute the fronts ▷ `Use fast non dominated sorting to divide` R_t `into` $F_1, F_2, \ldots$
  Find $i^* > 1$ such that $\sum_{i=1}^{i^*-1} |F_i| < N$ and $\sum_{i=1}^{i^*} |F_i| \geq N$, or $i^* = 1$ for $|F_1| \geq N$
  Calculate the crowding-distance() for each individual in $F_1, F_2, \ldots, F_{i^*}$
  Sort R_t based on Ranking and Crowding
  Let $\tilde{F}_{i^*}$ be the $N - \sum_{i=0}^{i^*-1} |F_i|$ individuals in F_{i^*} with largest crowding distance, chosen at random in case of a tie $P_{t+1} = \left(\cup_{i=1}^{i^*-1} F_i\right) \cup \tilde{F}_{i^*}$

it to find a set of solutions that are spread in the objective space and can provide convergence near the Pareto-optimal front.

NSGA-II (see Algorithm 3.7) starts with a random population (P_0) and divides this population into fronts (F) where the fronts contain the non-dominated individuals in the population. The individuals undergo evolutionary operators with respect to the parent selection and produce offspring. Next, two populations of initial and offspring are merged and divided into fronts, and their corresponding crowding distance of solutions for fronts is computed. The crowding distance is calculated as the average side length of the cuboid with respect to the nearest neighbours in the front of the solution. NSGA-II selects the next population, starting from including the highest-ranked front solutions to the inferior ones until it is full. We refer the reader to [Deb+02] for details on the crowding distance and fast non-dominated sort operators.

3.3.3 Particle Swarm Optimisation

Particle swarm optimisation (PSO) [KE95] is a meta-heuristic technique inspired by the social behaviour of birds and humans. Birds fly in flocks to find food sources and avoid predators. In human society, people learn from each other and they share knowledge from person to person to emerge a culture such as people in a living neighbourhood can learn locally from each other and share insights, and in this group learning, they can improve their social learning.

Each individual in PSO is a *particle* representing a potential solution in a population referred to as *swarm*. Particles move in the problem space and take advantage of the information they collect themselves and from their neighbours. Particles adjust their position with a random perturbation considering the best position visited before, and the best position found so far by their neighbours. With each iteration of the PSO system, the swarm moves toward areas with high-quality solutions.

We define the adjustment of position of particles as follows. Each particle has three vectors at the time (t) of evolution: position ($\vec{z_t}$), velocity ($\vec{v_t}$) and personal best, where it keeps the best position it has been evolved to as ($\vec{p_t}$). PSO updates particle positions based on the updated velocity for each component of the particles. c_1 and c_2 are cognitive and social factors that attract particles toward their personal

Algorithm 3.8: Particle Swarm Optimisation (PSO)

$t \leftarrow 0$
Initialise the position of particles ($\vec{z}_0$)
Initialise the velocity of particles ($\vec{v}_0$)
Calculate the quality of particles ($f(\vec{z}_0)$)
Set personal best of particles ($\vec{p}_0$) and global best ($\vec{p}_g$)
while *StopCriteria* $\neq$ *True* **do**
    $\vec{v}_{t+1} = \omega\vec{v}_t + c_1 r_1 \times (\vec{p}_t - \vec{z}_t) + c_2 r_2 \times (\vec{p}_g - \vec{z}_t)$ ▷ `Update velocity of particles`
    $\vec{z}_{t+1} = \vec{z}_t + \vec{v}_{t+1}$ ▷ `Update position of particles`
    Update personal best of particles
    Update global best
    $t \leftarrow t + 1$

Algorithm 3.9: Ant Colony Optimisation

Initialise the ants and initial pheromone
repeat
    foreach *ant* **do**
        repeat
            Choose one neighbour at each step probabilistically.
        until *solution is complete*
    foreach *selected ant* **do**
        Perform local search ▷ `Optional`
    Evaporate pheromones
    Deposit pheromones
until *termination criterion met*

and global best. r_1 and r_2 are vectors with size of a particle containing random values from a uniform distribution in the range [0, 1]. Algorithm 3.8 shows the procedure of PSO.

3.3.4 Ant Colony Optimisation

Another bio-inspired method draws inspiration from ants. Ants in nature take a random walk from their nest to locate a food source. On their way back to the nest, ants leave a substance called pheromone. Other ants can detect it and follow the favourable path for food. As more ants go along this path, the pheromone becomes stronger and more ants follow the favourable path. However, some pheromone progressively evaporates over time, resulting in reducing the attractiveness capability of unexplored routes.

Ant colony optimisation (ACO) [DS04] represents a mathematical platform employing artificial ants based on the foraging behaviour of ants. In nature, ants move asynchronously and lay pheromone while moving; however, in ACO, ants move synchronised and the lay pheromone after the trip. The pheromone value plays a key role in balancing the selection pressure. We employ the well-known Max-Min Ant System (MMAS) [SH00], which is one of many types of ACO in the literature. MMAS can limit the range of pheromones not to exceed a value where the pheromone plays a key role in the probabilistic selection at different steps of constructing a valid solution.

ACO has been successfully applied in combinatorial optimisation and open-pit mining scheduling problem [SS15; NSW09; CW20].

Algorithm 3.9 shows the procedure of ACO. ACO sets the initial value for the pheromone matrix (τ) for all edges in a combinatorial problem. Next, it generates artificial ants for its colony in the first generation. Each ant performs a random walk using the solution construction heuristic to generate a valid solution step by step. Employing the following probabilistic selection strategy with the probability of choosing the edge from i to j in a graph as follows.

$$p(e_{i,j}|\mathcal{N}) = \frac{[\tau_{i,j}]^{\alpha}[\eta(e_{i,j})]^{\beta}}{\sum_{e_{k,l}\in\mathcal{N}}[\tau_{k,l}]^{\alpha}[\eta(e_{k,l})]^{\beta}}$$

where α and β are ACO parameters that regulate the influence of heuristic (η) and pheromone information, respectively, the preceding equation implies that the selection of the next component in a solution is both dependent on the quality of edges in terms of the objective function and the pheromone information. $\mathcal{N}$ refers to the available possible components to select in the neighbourhood. The pheromone values get updated after all ants finalise their solution and a generation is completed.

Pheromone value plays an essential role in guiding ants to better solutions in further generations of the colony because it represents that if an edge has more pheromone, it means more ants have trespassed on that track which shows that the track could be profitable. There also exists an evaporating factor (ρ) to reduce attraction on reinforced routes and improve the exploration ability of artificial ants. ACO can integrate local search to improve the obtained solution more finely. ACO updates the pheromone matrix with the following general update rule.

$$\tau_{i,j} \leftarrow (1-\rho).\tau_{i,j} + \Delta\tau_{i,j}$$

where $0 < \rho \leq 1$ denotes the evaporation rate and $\Delta\tau_{i,j}$ refers to the effect of the selected ant to deposit pheromone which includes $e_{i,j}$ in its obtained solution.

3.4 Novelty Search

Novelty Search is a type of optimisation algorithm that is designed to discover and explore new, unique solutions to a given problem, rather than simply finding a best solution. It is based on the idea that innovation and creativity are important drivers of progress and that finding novel solutions can be just as important as finding optimal ones.

One of the key features of novelty search is that it does not rely on an explicit objective function or fitness measure to evaluate the quality of a solution. Instead, it uses a measure of novelty to identify solutions that are different from those that have been previously encountered [LS11a; LS08]. This allows novelty search to explore the search space more broadly and to discover solutions that may not be optimal according to the traditional evaluation metric, but that are still valuable in some way.

Novelty Search has been applied to a wide range of problems, including machine learning, evolutionary computation, robotics, and more. It has been shown to be particularly effective at solving problems with high-dimensional search spaces, where traditional optimization algorithms may struggle to find good solutions.

One of the main advantages of novelty search is that it can help to avoid the local optima problem, where an optimization algorithm gets stuck in a suboptimal region of the search space because it is unable to escape. By encouraging exploration of the

search space, novelty search can help to find globally optimal solutions rather than being trapped in a local optima.

Novelty Search has proved its capability in solving optimization problems in different contexts, as demonstrated by many studies in different contexts [LS11b]. despite its success in these areas, the application of Novelty Search to global and constrained optimization problems has been relatively limited. While there have been a few notable studies [Fis+19; Mar+19; LYT15] that have demonstrated the potential of Novelty Search in these contexts, there is still a significant opportunity for further research in this area.

Given the potential of Novelty Search to uncover solutions that traditional optimization methods may not find, it is important to continue to explore its use in global and constrained optimization problems. The optimization landscape in these contexts is often much more complex and challenging than in simpler problems, and therefore requires new approaches. By focusing on the application of Novelty Search to these types of problems, this thesis aims to make valuable contributions to the field and help to extend the boundaries of what is currently possible with this powerful algorithm.

Note that novelty search and quality-diversity search are both optimisation algorithms that are used to find solutions to complex problems. They are similar in that they both aim to find solutions that are novel or diverse, rather than simply optimising a single objective function. However, there are some key differences between the two approaches. Novelty search focuses on exploring the search space to find novel solutions, while quality-diversity search focuses on both exploring the search space and exploiting the best solutions found so far. Quality-diversity search aims to find a diverse set of high-quality solutions, while novelty search is primarily concerned with finding solutions that are novel, regardless of their quality. However, the relative performance of the two algorithms can depend on the specific problem being solved.

3.5 Conclusions

Throughout this chapter, we explained deterministic and randomised methods that we use later. We discussed how dynamic programming could solve the knapsack problem and how iterative local search could improve a solution iteratively. We presented the greedy algorithm. We then discussed evolutionary computation, including single and multi-objective evolutionary algorithms, particle swarm optimisation and ant colony optimisation. PSO, ACO, and EA all operate within this general framework and are used to search for an optimal solution in a given search space. However, the main differences between these algorithms lie in the specific methods and techniques used for selection, generation of offspring, and survival selection.

- EA uses genetic operations such as crossover and mutation to evolve the population over time. The selection of solutions is based on the fitness function, and the offspring are generated by applying genetic operations to the selected solutions. Survival selection is based on environmental selection methods such as elitist or truncation selection.

- PSO uses a population of particles that move in the search space guided by their own and their neighbours' best positions. The selection of solutions is based on the current position and velocity of the particles, and the offspring are generated by updating the position and velocity of each particle. The survival selection is based on the best position discovered so far by the swarm.

- ACO uses a population of ants that move to construct a solution through sampling from a probability distribution. Ants use this distribution of probability to make decisions about what components to take in the search space. The probability of this decision is influenced by factors such as the heuristic information about the quality of the solution and the presence of pheromones deposited by other ants. Best-obtained solutions are used to update the probabilities for future exploration.

It's important to note that these are just the basic descriptions of each algorithm, the actual implementations could have variations and specific features that sets them apart from others.

Chapter 4

Evolutionary Bi-objective Optimisation for the Dynamic Chance-Constrained Knapsack Problem Based on Tail Bound Objectives

Many real-world combinatorial optimisation problems involve stochastic as well as dynamic components which are mostly treated in isolation. However, in order to solve complex real-world problems, it is essential to treat stochastic and dynamic aspects in a holistic approach and to understand their interactions. Dynamic components in an optimisation problem may change the objective function, constraints or decision variables over time. The challenge to tackle a dynamic optimisation problem (DOP) is to track the moving optima when changes occur [NYB12]. Moreover, uncertainty is pervasive in a real-world optimisation problem. The source of uncertainty may involve the nature of data, measurement errors or lack of knowledge. Ignoring uncertainties in solving a problem may lead to obtaining suboptimal or infeasible solutions in practice [LL15].

In this chapter, we consider the chance-constrained knapsack problem (CCKP) which is described in Section 2.3 with dynamically changing constraint to which we refer to as dynamic chance-constrained knapsack problem (DCCKP). We also assume that each item in the knapsack problem has an uncertain weight while the profits are deterministic. For the dynamic component, we use the settings defined in [RNN18] and we follow the approach and settings proposed in [Xie+19] which employs inequality tails (see Section 2.3.1) to estimate the violation in probabilistic constraint.

Considering DCCKP, we aim to re-compute a solution of maximal profit after a dynamic change occurs to the capacity constraint, while the total uncertain weight can exceed the capacity with a small probability. We also take advantage of bi-objective optimisation. However, we cannot directly apply the second objective used in previous studies because they only considered either dynamic or stochastic aspect of the optimisation problem in isolation for the second objective function. To tackle this issue, we introduce an objective function which deals with the uncertainties and accommodate the dynamic aspects of the problems. This objective evaluates the smallest knapsack capacity bound for which a solution would not violate the chance constraint by making use of tail inequalities such as Chebyshev's inequality and Chernoff bounds to approximate the probabilistic constraint violation. The introduced objective also can keep a set of non-dominated solutions to be used for tracking the moving optimum.

For our investigation, we apply (1+1)-EA (see Algorithm 3.5), a modified variant

of GSEMO (see Algorithm 3.6) and NSGA-II (see Algorithm 3.7) to the problem where the last two compute the trade-offs with respect to the profit and the newly introduced objective for dealing with the chance constraints. We show that the bi-objective EAs perform better than the single objective approaches. Introducing the additional objective function to the problem helps the bi-objective optimisation algorithm to deal with the constraint changes as it obtains the non dominated solutions with respect to the objective functions.

This chapter is based on the work published in the European Conference on Artificial Intelligence 2020 [Ass+20]. We added more details into the partial offline error calculation as well as adding more experiments considering bigger knapsack instances. The rest of this chapter is organised as follows. In the next section, we define the DCCKP and how to assess the performance of the algorithms. Next, we introduce the objective function for dealing with DCCKP and develop the bi-objective model. Subsequently, we report on the behaviour of single objective and bi-objective baseline EAs in solving DCCKP followed by some concluding remarks.

4.1 Dynamic Chance-Constrained Knapsack Problem

Recall the definition of chance-constrained knapsack problem in Section 2.3, we assume that the weight of each item is chosen independently according to a given probability distribution. We also consider that the knapsack capacity is dynamic and changes over time every τ iterations and we call τ the frequency of changes, which denotes after how many iterations a change occurs in the knapsack capacity with the magnitude of changes as r according to some probability distributions. For example, when τ is 1000 and r is 500, the knapsack capacity is changing every 1000 iterations with a magnitude of change as 500.

We integrate the stochastic and dynamic components of the problem that we aim to find a subset of items with stochastic weight such that the probability of exceeding knapsack capacity is bound by α and the knapsack capacity changes over time. We use the binary knapsack test problems introduced in [Pol+14] and later developed for dynamic knapsack optimisation in [RNN18]. In the latter, they set the environment to let the algorithms have τ iterations window to find the optimum for the capacity in that moment and get prepared for the next change in the capacity. The combination of low and high frequencies bring about different types of challenges for the algorithm where with a high frequency gives less time for the algorithm to adapt to the changes and if this time has been solely spent on one single solution (such as the solution in (1+1)-EA) there could be higher improvement rather than using an EA with a population.

Recall different types of knapsack problem described in Section 2.2, we consider two types of *uncorrelated* and *bounded strongly correlated* test problems. The latter is more difficult to solve because the profit correlates with the weight [Pol+14]. Note that in our dynamic chance-constrained setting, for bounded strongly correlated instances, we consider the correlation between the expected weight and the profit.

We present the dynamic parameters for our investigation as follows. We define r which determines the magnitude of changes. We consider changes according to the uniform distribution in $[-r, r]$ where $r \in \{500, 2000\}$ to consider the small and large magnitude of changes in the knapsack constraint, respectively. We also set $\tau \in \{100, 1000\}$ to observe fast and slow changes in the constraint, respectively. For the stochastic parameters, we set the parameters according to [Xie+19]. Therefore we set $\alpha \in \{0.01, 0.001, 0.0001\}$ to consider different values of restriction for the

probability of chance-constraint violation probability. These small values represent the reliability in a design where there are 99%, 99.9%, and 99.99% reliability, respectively. These metrics are designed to evaluate the likelihood of constraints being violated to varying degrees. We also set $\delta \in \{25, 50\}$ to assign small and large uncertainty interval in the weight of items with uniform distribution. To ensure that the weights of items subject to uncertainty are positive, we also add a value of 100 to all the weights to avoid negative values.

We use dynamic programming (see Algorithm 3.2) to find the exact optimal solution with maximal profit $P(x^*)$ for the deterministic variant of the knapsack problem. Therefore, we record $P(x^*)$ for every dynamic capacity change for each knapsack instances based on r and τ.

To evaluate the performance of our algorithms for DCCKP, we consider the offline error which represents the distance between the algorithm's best-obtained solution in each iteration with respect to $P(x^*)$. Let x be the best solution obtained by the considered algorithm in iteration i. The offline error for iteration i is given as

$$\phi_i = \begin{cases} P(x^*) - P(x) & \text{if } \Pr[W(x) \geq C] \leq \alpha \\ (1 + \mathit{penalty}(x))\, P(x^*) & \text{otherwise.} \end{cases}$$

It is important to note that the offline error calculation is viewed from the perspective of an algorithm, where the inequality tail is integrated with an algorithm to determine whether a solution is feasible or not. Furthermore, we can also use newly introduced second objective for offline error calculation analogously. However, we define it based on the chance-constrained probability because we are considering single-objective algorithms in our comparisons.

In addition, we should take into account of the fact that the calculation of probabilistic term ($\Pr[W(x) \geq C]$) can only be done if the total expected weight ($\mathbb{E}(W(x))$ is less than the knapsack bound. For instances where the total expected weight exceeds the knapsack bound, we use a corresponding *penalty* factor with is proportionate to the degree of violation of feasibility.

We define the *penalty* as follows.

$$\mathit{penalty} = \begin{cases} \Pr[W(x) \geq C] & \text{if } \Pr[W(x) \geq C] > \alpha \\ 1 + (\mathbb{E}(W(x)) - C) & \text{if } C < \mathbb{E}(W(x)) \end{cases}$$

Every solution x not meeting the chance constraint receives a higher offline error than any solution meeting the chance constraint. The total offline error is the summation of offline error at each iteration divided by the number of total iterations ($maxiter$) as follows.

$$\Phi = \frac{\sum_{i=1}^{maxiter} \phi_i}{maxiter}.$$

4.2 Bi-Objective Optimisation Model

It has been shown that transforming a single-objective optimisation problem into bi-objective by adding a conflicting objective can lead to obtaining better solutions (see Section 2.5). Therefore for DCCKP, we introduce an additional objective which estimates the minimal capacity bound for a given stochastic solution that still meets the

chance constraint. This objective helps to cater for dynamic changes to the stochastic problem.

We redefine DCCKP by introducing a new second objective function to transform it into a bi-objective optimisation problem. Therefore, we introduce (C^*) as the stochastic bound as our second objective function. This objective function evaluates the smallest knapsack capacity for a given solution such that it satisfies the predefined limit on the chance constraint. Therefore, the fitness $f(x)$ of a solution x is given as

$$f(x) = (P(x), C^*(x))$$

where

$$C^*(x) = \min \{C \mid \text{s.t. } \Pr[W(x) \geq C] \leq \alpha\}$$

is the smallest weight bound C such that the probability that the weight $W(x)$ of x is at least C is at most α. Using this objective allows to cater for dynamic changes of the weight bound of our problem. In bi-objective optimisation of DCCKP, the goal is to maximise $P(x)$ and minimise $C^*(x)$. Hence, we have

$$f(x') \succeq f(x) \text{ iff } P(x') \geq P(x) \wedge C^*(x') \leq C^*(x)$$

for the dominance (see Section 2.5) relation of bi-objective optimisation for two solutions namely x and x'.

Evaluating the chance constraint is computationally difficult [BGN09]. It has been shown that even if random variables are from a Bernoulli distribution, calculating the probability of violating the constraint exactly is #P-complete, see Theorem 2.1 in [KRT00]. Because it is difficult to compute C^* exactly, we make use of the tail inequalities described in the previous chapter to calculate the second objective function. For Chebyshev's inequality, the stochastic bound is given as follows.

Proposition 1 (Chebyshev Constraint Bound Calculation)**.** *Let $\mathbb{E}(W(x))$ be the expected weight, $\sigma^2_{W(x)}$ be the variance of the weight of solution x and α be the probability bound of the chance constraint. Then setting $C^*_1(x) = \mathbb{E}(W(x)) + \sigma_{W(x)}\sqrt{\frac{1-\alpha}{\alpha}}$ implies $\Pr[W(x) \geq C^*_1(x)] \leq \alpha$.*

Proof. Using Chebyshev's inequality (see Section 2.3), we have

$$\Pr\left[W(x) \geq \mathbb{E}(W(x)) + \lambda\right] \leq \frac{\sigma^2_{W(x)}}{\sigma^2_{W(x)} + \lambda^2}.$$

We set $C^*_1(x) = \mathbb{E}(W(x)) + \lambda$ which implies

$$\frac{\sigma^2_{W(x)}}{\sigma^2_{W(x)} + \lambda^2} = \alpha$$

Therefore we have

$$\begin{aligned} \lambda &= \sqrt{\frac{\sigma^2_{W(x)}(1-\alpha)}{\alpha}}, \\ &= \frac{\sigma_{W(x)}\sqrt{\alpha(1-\alpha)}}{\alpha}, \\ &= \sigma_{W(x)}\sqrt{\frac{(1-\alpha)}{\alpha}}. \end{aligned}$$

Hence, we have

$$\begin{aligned}
&\Pr[W(x) \geq C_1^*(x)] \\
&= \Pr\left[W(x) \geq \mathbb{E}(W(x) + \sigma_{W(x)}\sqrt{\tfrac{(1-\alpha)}{\alpha}}\right] \\
&\leq \frac{\sigma^2_{W(x)}}{\sigma^2_{W(x)} + \left(\sigma_{W(x)}\sqrt{\frac{(1-\alpha)}{\alpha}}\right)^2} \\
&= \alpha
\end{aligned}$$

which completes the proof. □

We consider $w_i \in \mathcal{U}[\mathbb{E}(w_i) - \delta, \mathbb{E}(w_i) + \delta]$ and $\sum_{i=1}^{n} x_i$ denotes the total number of chosen items in a solution. We know for a uniform distribution that,

$$\sigma_{W(x)} = \delta\sqrt{\frac{\sum_{i=1}^{n} x_i}{3}}.$$

We substitute λ as

$$\begin{aligned}
\lambda &= \delta\sqrt{\tfrac{\sum_{i=1}^{n} x_i}{3}}\sqrt{\tfrac{(1-\alpha)}{\alpha}}, \\
&= \tfrac{\delta\sqrt{3\alpha(1-\alpha)\sum_{i=1}^{n} x_i}}{3\alpha}.
\end{aligned}$$

then the stochastic bound based on Chebyshev's inequality for uniform distribution is given as follows. Therefore, we have

$$C_1^*(x) = \mathbb{E}(W(x)) + \frac{\delta\sqrt{3\alpha(1-\alpha)\sum_{i=1}^{n} x_i}}{3\alpha}. \tag{4.1}$$

Moreover, to derive the second objective function by making use of the Chernoff bound, we have:

Proposition 2 (Chernoff Constraint Bound Calculation)**.** *Let $w_i \in \mathcal{U}[\mathbb{E}(w_i)-\delta, \mathbb{E}(w_i)+\delta]$ be independent weights chosen uniformly at random. Let $\mathbb{E}(W(x))$ be the expected weight of x and α be the probability bound of the chance constraint. Then setting*

$$\begin{aligned}
&C_2^*(x) = \mathbb{E}(W(x)) \\
&-0.66\delta\left(\ln(\alpha) - \sqrt{\ln^2(\alpha) - 9\ln(\alpha)\sum_{i=1}^{n} x_i}\right)
\end{aligned} \tag{4.2}$$

implies $Pr[W(x) \geq C_2^*(x)) \leq \alpha$.

Proof. We consider $w_i \in \mathcal{U}[\mathbb{E}(w_i) - \delta, \mathbb{E}(w_i) + \delta]$. Then, to satisfy Chernoff bound summation requirement, we normalise each random weight into [0, 1] which y_i denotes the normalised weight of the knapsack item as follows.

$$y_i = \frac{w_i - (\mathbb{E}(w_i) - \delta)}{2\delta} \in [0, 1]$$

$$Y(x) = \sum_{i=1}^{n} y_i = \sum_{i=1}^{n} \frac{w_i - (\mathbb{E}(w_i) - \delta)}{2\delta} x_i.$$

Since y_i is symmetric, then the total expected weight of Y is in the middle as $\mathbb{E}(Y) = \frac{1}{2}\sum_{i=1}^{n} x_i$. Then, the total weight of a solution is given as

$$W(x) = \sum_{i=1}^{n} w_i x_i = 2\delta Y(x) + \mathbb{E}(W(x)) - \delta\sum_{i=1}^{n} x_i.$$

We set

$$C_2^* = \mathbb{E}(W(x)) + b$$

where

$$b = -0.66\delta \left(\ln(\alpha) - \sqrt{\ln^2(\alpha) - 9\ln(\alpha)\sum_{i=1}^{n} x_i} \right).$$

Hence, the probability of violating the chance constraint for a solution is given as

$$\begin{aligned} & \Pr\left[W(x) \geq C_2^*(x)\right] \\ = \; & \Pr\left[2\delta Y(x) + \mathbb{E}(W(x)) - \delta\textstyle\sum_{i=1}^{n} x_i \geq \mathbb{E}(W(x)) + b\right] \\ = \; & \Pr\left[Y(x) \geq \tfrac{1}{2}\textstyle\sum_{i=1}^{n} x_i + \frac{b}{2\delta}\right] \\ = \; & \Pr\left[Y(x) \geq \mathbb{E}\left(Y(x)\right) + \frac{b}{2\delta}\right] \\ = \; & \Pr\left[Y(x) \geq (1+t)\mathbb{E}\left(Y(x)\right)\right] \end{aligned}$$

where

$$t = \frac{b}{2\delta\mathbb{E}\left(Y(x)\right)}.$$

Using Chernoff bounds, we have

$$\Pr\left[(Y(x) \geq (1+t)\mathbb{E}\left(Y(x)\right))\right] \leq \exp\left(-\frac{t^2}{2+\frac{2}{3}t}\mathbb{E}(Y(x))\right)$$

We have

$$t = \frac{-0.66\left(\ln(\alpha) - \sqrt{\ln^2(\alpha) - 9\ln(\alpha)\sum_{i=1}^{n} x_i}\right)}{\sum_{i=1}^{n} x_i}.$$

That we can expand as,

$$t = \frac{-0.66\ln(\alpha)}{\sum_{i=1}^{n} x_i} + 0.66\frac{\sqrt{\ln^2(\alpha) - 9\ln(\alpha)\sum_{i=1}^{n} x_i}}{\sum_{i=1}^{n} x_i}$$

To find an upper bound for this equation we use

$$\hat{t} = \frac{-0.66\ln(\alpha)}{\sum_{i=1}^{n} x_i} \leq \frac{b}{2\delta\mathbb{E}(Y(x))} = t$$

instead of t which results in

$$\begin{aligned} & \Pr\left[Y(x) \geq (1+t)\mathbb{E}\left(Y(x)\right)\right] \\ \leq \; & \Pr\left[Y(x) \geq (1+\hat{t})\mathbb{E}\left(Y(x)\right)\right] \\ \leq \; & \exp\left(-\frac{\frac{(-0.66)^2\ln^2\alpha}{(2\mathbb{E}(Y(x)))^2}}{2+\frac{2}{3}\left(\frac{-0.66\ln\alpha}{2\mathbb{E}(Y(x))}\right)} \cdot \mathbb{E}\left(Y(x)\right)\right) \\ = \; & \exp\left(\frac{\ln\alpha}{\frac{-8\mathbb{E}(Y(x))+\frac{4}{3}(0.66\ln\alpha)}{(-0.66)^2\cdot\ln\alpha}}\right) \\ \leq \; & \alpha \end{aligned}$$

Therefore, we have

$$\frac{-4\sum_{i=1}^{n} x_i + \frac{4}{3}(0.66\ln\alpha)}{(0.66)^2\ln\alpha} \leq 1$$

This inequality holds as

$$-4\sum_{i=1}^{n} x_i + \frac{4}{3}(0.66 \ln \alpha) \leq (0.66)^2 \ln \alpha$$

which completes the proof. □

Note that the introduced additional objectives as C_1^* and C_2^* in Equations 4.1 and 4.2, respectively calculate the smallest possible bound for which a solution meets the chance constraint according to the used tail bound (Chebyshev or Chernoff). The terms are added to the expected total weight guarantee that a given solution meets the chance constraint.

4.3 Evolutionary Algorithms

In this section, we discuss the use of EAs to solve the DCCKP. We only use simple baseline EAs to make a fair comparison between the single-objective optimisation and bi-objective optimisation. (1+1)-EA (see Algorithm 3.5) and GSEMO (see Algorithm 3.6) are their equivalent counterparts if we consider identical objective functions because they use the same mutation operator.

4.3.1 Single-Objective Optimisation

(1+1)-EA has one potential solution (x) that undergoes mutation and offspring x' replaces x if it is determined to be at least as good or better according to the fitness function of a solution which is as follows,

$$f_{(1+1)}(x) = (\max\{0, \alpha(x) - \alpha\}, P(x))$$

Recall that from Section 2.3 that $\alpha(x)$ denotes the probability of chance constraint violation based on Chebyshev's inequality or Chernoff bound derived for CCKP where for uniform distribution we use Equations 2.2 and 2.3 for Chebyshev's inequality and Chernoff's bound, respectively.

The fitness function $f_{(1+1)}$ is in lexicographic order which means that first, the algorithm searches for a feasible solution according to the chance constraint and maximises the profit afterwards. We have,

$$\begin{aligned} & f_{(1+1)}(x') \succeq f_{(1+1)}(x) \\ \iff\ & (\max\{0, \alpha(x') - \alpha\} < \max\{0, \alpha(x) - \alpha\}) \\ & \vee\ ((\max\{0, \alpha(x') - \alpha\} = \max\{0, \alpha(x) - \alpha\}) \\ & \wedge (P(x') \geq P(x))) \end{aligned}$$

When a change occurs in the dynamic constraint, the individual (x) may become infeasible, and its probabilistic constraint violates α. Therefore, (1+1)-EA mutates x to find a feasible solution for the newly given constraint and optimises the profit afterwards.

4.3.2 Bi-Objective Optimisation

We adapt the algorithm proposed in [RNN18] for our bi-objective optimisation. We call the adapted algorithm, Pareto Optimisation for Stochastic Dynamic Constraint (POSDC). POSDC (see Algorithm 4.1) is a baseline multi-objective EA which tracks

Algorithm 4.1: POSDC

```
Generate x ∈ {0,1}^n uniformly at random
if C − η ≤ C*(x) ≤ C + η then
    S ← x
else
    while S = ∅ do
        repair an offspring (y) by (1+1)-EA
        x ← y
        if C − η ≤ C*(y) ≤ C + η then
            S ← x
while (not max iteration) do
    if change in the capacity occurs (after τ iterations) then
        x ← best solution in S
        Update S⁻ and S⁺ with respect to the shifted capacity
        if S = ∅ then
            S ← x
    choose x ∈ S uniformly at random
    y ← create an offspring by flipping each bit of x independently with the
      probability of 1/n
    if (C − η ≤ C*(y) < C) ∧ (∄z ∈ S⁻ : z ⪰_POSDC y) then
        S⁻ ← (S⁻ ∪ y) \ {z ∈ S⁻ | y ⪰_POSDC z}
    else if (C ≤ C*(y) ≤ C + η) ∧ (∄z ∈ S⁺ : z ⪰_POSDC y) then
        S⁺ ← (S⁺ ∪ y) \ {z ∈ S⁺ | y ⪰_POSDC z}
return best solution
```

the moving optimum by storing a population in the vicinity of the dynamic knapsack capacity. POSDC keeps a solution (x) if $C^*(x)$ is in $[C-\eta, C+\eta]$, where η determines the storing range. Therefore, POSDC has two subpopulations which include feasible and infeasible solutions ($S = S^- \cup S^+$). Keeping an infeasible subpopulation helps POSDC to be prepared for the next change in the dynamic constraint. We define these two subpopulations as follows.

$$S^- \leftarrow \{x \in S \mid C - \eta \leq C^*(x) \leq C\}$$
$$S^+ \leftarrow \{x \in S \mid C < C^*(x) \leq C + \eta\}.$$

POSDC generates the initial solution uniformly at random, if the generated solution is out of the storing range, then (1+1)-EA repairs the solution and stores it in the appropriate subpopulation.

POSDC uses a mutation operator to explore the search space and find trade-off solutions. POSDC maintains a set of non-dominated solutions with respect to $P(x)$ and $C^*(x)$ in its subpopulations. The best solution in POSDC at each iteration is the solution with the highest profit in S^-; If S^- is empty, POSDC prefers the solution with the smallest C^* in S^+.

Note that if we can compute the solutions exactly, some solutions in S^+ can be feasible. However, because computing C^* in exact is difficult, we designate the optimum as the solution with the highest profit in S^-. Also, the parameter η for POSDC has been considered equal to r to cover the interval of the uniform distribution entirely for storing desirable solutions [RNN18].

For further investigation of our bi-objective optimisation, we also apply NSGA-II

to point out possible improvements by using a well-established method for bi-objective optimisation. We modify NSGA-II to keep the best-obtained solution for the given knapsack bound C in each iteration to have a fair comparison with POSDC. The main difference between NSGA-II and POSDC is the selection mechanism.

4.4 Experimental Investigation

In this section, we define the setup of our experimental investigation. We apply the bi-objective optimisation with the introduced objectives and compare it with the single-objective optimisation.

The implemented framework is available on `https://bit.ly/3w0fFZJ`. This repository includes the implementation of (1+1)-EA, POSDC and NSGA-II for the dynamic chance-constrained knapsack problem. The source code language is Python, and there is a *run* file for each algorithm to simulate the experiments. The repository also includes information on the dynamic changes of the knapsack capacity and the knapsack instance for 100, 300 and 500. Each algorithm in the framework includes instructions on how to run.

We combine the parameters of r, τ, α and, δ to produce DCCKP test problem instances for uncorrelated and bounded strongly correlated with different types of complexities. For instance, a test problem with $r = 2000$, $\tau = 100$, $\alpha = 0.0001$ and $\delta = 50$ represents the most difficult test problem, because the magnitude of dynamic change in the knapsack capacity is large and the capacity changes very fast every 100 iterations. Also, the allowable probability of chance-constraint violation is very tight, and the uncertainty interval in the weight of items is big.

We apply POSDC and (1+1)-EA integrated with Chebyshev and Chernoff inequality tails to DCCKP instances. Specifically, we investigate the following algorithms:

- (1+1)-EA with Chebyshev's inequality: (1)
- (1+1)-EA with Chernoff bound: (2)
- POSDC with Chebyshev's inequality: (3)
- POSDC with Chernoff bound: (4)
- NSGA-II with Chebyshev's inequality: (5)
- NSGA-II with Chernoff bound: (6)

Each algorithm initially runs for 10^4 warm-up iterations before the first change in the capacity occurs and continues for 10^6 iterations.

Tables 4.1-4.9 report the performance of single-objective and bi-objective optimisation by the average and standard deviation of total offline error for 30 independent runs. Lower total offline error is better because it shows the algorithm was closer to the $P(x^*)$ for each iteration. Note that when the problem becomes more uncertain, the feasible region (without violating the probabilistic constraint) becomes more restrictive and the offline error will be increased.

Statistical comparisons are carried out by using the Kruskal-Wallis test with 95% confidence interval integrated with the posteriori Bonferroni test to compare multiple solutions [CF14]. The stat column shows the rank of each algorithm in the instances. If two algorithms can be compared with each other significantly, $X^{(+)}$ denotes that the current algorithm is outperforming algorithm X. Likewise, $X^{(-)}$ signifies the current algorithm is worse than the algorithm X significantly. Otherwise, $X^{(*)}$ shows that the

current algorithm is not different significantly with algorithm X. For example, numbers $1^{(+)}, 3^{(*)}, 4^{(-)}$ denote the pairwise performance of algorithm (2). The numbers show that algorithm (2) is statistically better than algorithm (1); it is not different from algorithm (3) and it is inferior to algorithm (4).

Note that statistical tests are commonly used, but proper understanding of their conclusions is important. A test can show that differences are statistically significant or not, but it doesn't imply difference in algorithm performance. The test is inconclusive if the p-value is larger than the chosen confidence level.

Table 4.1 lists the results when r is 500 and n is 100. We can see that for the uncorrelated instances when *alpha* is the largest (0.01) and the interval of uncertainty is small, (1+1)-EA-Chebyshev and (1+1)-EA-Chernoff are not significantly different. However, (1+1)-EA-Chernoff outperforms the (1+1)-EA-Chebyshev when there is more time to adapt ($\tau = 1000$) to the dynamic change and the uncertainty interval is the largest. We can observe the same for bounded-strongly-correlated instances where for all instances when α is 0.01, both (1+1)-EA variants are not significantly different.

In all other instances, (1+1)-EA-Chernoff outperforms another variant for both uncorrelated and bounded-strongly-correlated instances. Alike to (1+1)-EA variant, POSDC variants also show no significant difference when α is 0.01 except when τ is 1000 and the δ is 50. POSDC-Chebyshev outperforms its single objective variant in 9 and 8 instances for uncorrelated and bounded-strongly-correlated instances. Similarly, POSDC-Chernoff outperforms its (1+1)-EA variant in all instances for both types of problems.

Table 4.2 lists our results when r increases to 2000. We can see that when τ is the largest, there is no significant difference between two (1+1)-EA variants. However, as the problem becomes more complex, (1+1)-EA-Chebyshev outperforms the other except when (1+1)-EA-Chernoff has a bigger time window to adapt even though the chance constraint is too tight and the uncertainty interval is the biggest. Interestingly, in this instance, (1+1)-EA-Chernoff can outperform POSDC-Chebyshev. We can see that POSDC-Chebyshev can outperform both (1+1)-EA variants in other instances.

Between POSDC variants, there is not a uniform pattern to interpret. We can see that both variants can outperform the another in 6 separate instances. We observe that POSDC can obtain better solutions than (1+1)-EA. When we consider a bigger magnitude of changes in the constraint bound, the population size of non-dominated solutions in POSDC is bigger than when r is 500; because η is equal to r, POSDC covers a bigger range of solutions which leads to a bigger population.

Therefore, when the changes occur faster (smaller τ), POSDC has less time to evolve its population. POSDC only mutates one individual chosen randomly in its population, leading to a lower chance of choosing the best individual for the mutation in its population. In contrast, (1+1)-EA only handles one individual, mutates and improves it on all iterations. Introducing our second objective function for the bi-objective optimisation approach helps POSDC to tackle all these drawbacks and outperform its counterpart single-objective approach; because trade-off solutions contain more information in the principle of finding better solutions. Comparing POSDC-Chebyshev and POSDC-Chernoff we can see that the former can outperform the other one in the most complex condition. However, when the algorithm has more time to adapt, POSDC-Chebyshev is the best in particular when the α gets tighter.

To further investigate our bi-objective optimisation, we also apply the NSGA-II, a state of the art multi-objective EA, when dealing with two objectives. We run NSGA-II with a population size of 20 using Chebyshev and Chernoff inequality tails (algorithms (5) and (6), respectively, in corresponding tables for NSGA-II.

Table 4.3 shows the results of NSGA-II with small and large changes for uncorrelated and bounded strongly correlated instances when $n = 100$ and compares the performance of NSGA-II with POSDC. We can see that for both types of problems, NSGA variants can perform as well as POSDC when using the same tail inequality and when the change of magnitude in dynamic knapsack capacity is smaller, POSDC-Chernoff can outperform the NSGA-II-Chebyshev in 10 and 9 instances when r is 500 for uncorrelated and strongly-bounded-correlated instances, respectively. However, when r is 2000, NSGA-II-Chebyshev is weaker than POSDC-Chernoff in 2 and 4 instances, respectively, for uncorrelated and strongly-bounded-correlated instances.

Comparing two variants of NSGA-II, we can see that when instances are uncorrelated and r is 500, NSGA-II-Chernoff can outperform the other except when τ is 100 δ is 25, where they are not significantly different. The same behaviour can be seen for instances when r is 2000, and both variants are insignificant when α is the largest irrespective of the frequency.

Table 4.4 shows the results for POSDC and (1+1)-EA variant for $r = 500$ when number of items are 300. We can see that (1+1)-EA-Chernoff performs better or as well as another variant with Chebyshev's inequality tail in all uncorrelated instances. Moreover, (1+1)-EA-Chernoff outperforms (1+1)-EA-Chebyshev in all bounded-strongly-correlated instances. Comparing (1+1)-EA with the bi-objective counterpart algorithm when they use the same inequality tail, we can see that the POSDC variant can outperform the single-objective algorithm in all uncorrelated instances. The same observation can be seen for bounded-strongly-correlated instances except in one case where both algorithms are not significantly different. Comparing both POSDC variants, we can see that POSDC-Chernoff is the best or perform as well as the another in all instances.

Table 4.5 shows the results for POSDC and (1+1)-EA variant for $r = 2000$ when number of items are 300. Comparing (1+1)-EA variants, we can see that (1+1)-EA-Chebyshev can perform better or as well as another single-objective variant in all instances. The previous observation holds that the POSDC variant with the same inequality tail outperforms the single-objective counterpart. Comparing POSDC variants, we can see that both variants are not significantly different for 7 instances in each type of problem. However, in other instances, mostly POSDC-Chebyshev is the better algorithm.

Table 4.6 shows the results of NSGA-II with small and large changes for uncorrelated and bounded strongly correlated instances when $n = 300$ and compares the performance of NSGA-II with POSDC. We can see that when r is 500, NSGA-II variants can outperform their POSDC counterpart when they employ the same inequality tail. However, comparing two NSGA-II variants with each other, we see that NSGA-II-Chernoff is the best. When the problem gets more complex as r increases to 2000, when the dynamic capacity changes faster, both NSGA-II variants are not significantly different in all cases except one. However, when the changes occur slowly, NSGA-II-Chernoff is the best algorithm.

Table 4.7 shows the comparison between (1+1)-EA and POSDC variant for when the number of items is 500, and r is 500. Comparing (1+1)-EA variants, we can see that between (1+1)-EA variants, (1+1)-EA-Chernoff is performing better or as well as the another for all instances. Comparing single objective algorithms with their corresponding POSDC counterparts using the same inequality, we find that bi-objective optimisation outperforms the another for all instances. Comparing the POSDC variants, POSDC-Chernoff outperforms in most instances and performs as well as the another in two instances where α is 0.01.

Table 4.8 lists the comparison for when r is 2000 where we can see between (1+1)-EA variants, (1+1)-EA Chebyshev performs better or as well as the another. Comparing POSDC with (1+1)-EA, both POSDC variants outperform the single-objective counterpart. However, comparing POSDC variants, POSDC-Chernoff outperforms or performs as well as the other in most cases (9 out of 12) for each problem type. However, there is no clear pattern in the behaviour of the algorithm.

Table 4.9 shows the comparison between NSGA-II and POSDC variants. We can see that when the r is 500, NSGA-II Chernoff outperforms NSGA-II-Chebyshev for all instances. However, when r increases, for uncorrelated instances, NSGA-II Chebyshev works better or as well as the other when the change occurs faster. But when the changes are slower, NSGA-II Chernoff is the better algorithm. For bounded-strongly-correlated instances, in most settings, NSGA-II-Chernoff is the better algorithm. The main difference between NSGA-II and POSDC is the selection mechanism.

Summing up the findings for all instances irrespective of the number of items, we observe that when the environment of the problem becomes more complex, finding a solution that has a close distance to the optimal solution is harder. As τ decreases, δ increases and α becomes tighter, the offline error for both (1+1)-EA and POSDC increases. However, as the problem becomes more challenging to solve, POSDC obtains solutions with a lower total offline error than (1+1)-EA. Comparing POSDC and NSGA-II, the latter uses the crowding distance sorting to maintain its diversity through the evolution of its population. This comparison can point out the possible research direction to further investigate the state-of-art non-baseline EAs and multi-objective EAs solving DCCKPs.

4.5 Conclusions

In this chapter, we considered the dynamic chance-constrained knapsack problem where the constraint bound changes dynamically over time, and item weights are uncertain. The key part of our approach is to tackle the dynamic and stochastic components of an optimisation problem in a holistic approach. For this purpose and to apply bi-objective optimisation to the problem, we developed an objective C^* which calculates for a given solution x the smallest possible bound for which x would meet the chance constraint. This objective function allows keeping a set of non-dominated solutions with different C^* where an appropriate solution can be used to track the optimum after the dynamic constraint bound has changed. As it is hard to calculate the bound $C^*(x)$ in the stochastic setting exactly, we have shown how to calculate upper bounds for $C^*(x)$ based on Chernoff bound and Chebyshev's inequality. We evaluated the bi-objective optimisation for a wide range of chance-constrained knapsack problems with dynamically changing constraint bounds. The results show that the bi-objective optimisation with the introduced additional objective function can obtain better results than single-objective optimisation in most cases. Note that we also applied NSGA-II to the problem to point out possible improvements by using state of the art algorithms. It would be interesting for future work to extend these investigations. In addition, our approach is not limited to dynamic chance-constrained knapsack problems and the formulation can be adapted to a wide range of other problems where we would formulate a similar second objective to deal with the chance constraint.

TABLE 4.1. Statistical results of total offline error for (1+1)-EA and POSDC with small change ($r = 500$) in the dynamic constraint with $n = 100$

	τ	δ	α	(1+1)-EA-Chebyshev (1)			(1+1)-EA-Chernoff (2)			POSDC-Chebyshev (3)			POSDC-Chernoff (4)		
				Mean	Std	Stat	Mean	Std	Stat	Mean	Std	Stat	Mean	Std	Stat
uncorrelated	100	25	0.01	5745.77	1697.26	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	5551.26	1465.12	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	1785.78	459.47	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	1665.71	483.12	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	100	25	0.001	6047.05	642.79	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	5073.43	929.29	$1^{(+)}$, $3^{(-)}$, $4^{(-)}$	3155.38	393.46	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	1639.78	294.41	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	100	25	0.0001	10261.56	964.08	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	4905.78	694.76	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	7851.29	798.16	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	1733.7	258.58	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	100	50	0.01	5379.3	671.71	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	5138.64	639.58	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	2283.86	286.64	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	2069.15	292.86	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	100	50	0.001	8096.07	794.29	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	5348.09	613.04	$1^{(+)}$, $3^{(*)}$, $4^{(-)}$	5429.11	622.47	$1^{(+)}$, $2^{(*)}$, $4^{(-)}$	2410.94	351.49	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	100	50	0.0001	14744.88	1595.59	$2^{(-)}$, $3^{(*)}$, $4^{(-)}$	5583.39	630.6	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	13479.59	1278.83	$1^{(*)}$, $2^{(-)}$, $4^{(-)}$	2731.23	403.29	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	25	0.01	2729.05	172.84	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	2636.07	181.19	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	996.95	58.0	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	900.01	82.42	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	1000	25	0.001	4297.22	284.87	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	2686.15	88.86	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	2896.05	136.62	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	1126.39	103.46	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	25	0.0001	8555.38	1112.06	$2^{(-)}$, $3^{(*)}$, $4^{(-)}$	2851.67	119.43	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	7525.96	817.17	$1^{(*)}$, $2^{(-)}$, $4^{(-)}$	1331.85	126.84	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	50	0.01	3377.38	165.98	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	3211.46	137.9	$1^{(+)}$, $3^{(-)}$, $4^{(-)}$	1894.85	89.0	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	1719.41	130.66	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	50	0.001	6498.57	607.9	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	3620.4	165.19	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	5281.14	379.37	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	2161.92	167.01	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	50	0.0001	12024.57	2423.49	$2^{(-)}$, $3^{(*)}$, $4^{(-)}$	3983.41	191.37	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	11654.42	2124.35	$1^{(*)}$, $2^{(-)}$, $4^{(-)}$	2555.97	199.56	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
bounded-strongly-correlated	100	25	0.01	4046.88	399.33	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	3956.88	414.2	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	1710.77	161.25	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	1576.09	142.7	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	100	25	0.001	5266.34	759.26	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	3816.54	207.51	$1^{(+)}$, $3^{(-)}$, $4^{(-)}$	3231.89	447.91	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	1621.14	103.98	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	100	25	0.0001	9797.9	2112.68	$2^{(-)}$, $3^{(*)}$, $4^{(-)}$	3812.65	298.26	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	7847.42	1453.15	$1^{(*)}$, $2^{(-)}$, $4^{(-)}$	1755.29	140.92	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	100	50	0.01	4414.07	469.07	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	4147.15	418.58	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	2340.06	266.75	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	2101.0	204.38	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	100	50	0.001	7539.47	1385.14	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	4456.28	523.56	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	5501.32	897.8	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	2458.55	248.25	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	100	50	0.0001	14110.7	4032.99	$2^{(-)}$, $3^{(*)}$, $4^{(-)}$	4746.05	608.44	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	12931.85	3141.66	$1^{(*)}$, $2^{(-)}$, $4^{(-)}$	2787.91	286.62	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	25	0.01	2030.55	179.17	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	1935.4	159.31	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	809.06	86.87	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	727.53	60.2	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	1000	25	0.001	3517.55	494.67	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	2082.91	218.29	$1^{(+)}$, $3^{(*)}$, $4^{(-)}$	2263.7	283.82	$1^{(+)}$, $2^{(*)}$, $4^{(-)}$	904.61	70.71	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	25	0.0001	6990.66	1330.77	$2^{(-)}$, $3^{(*)}$, $4^{(-)}$	2227.49	237.11	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	5707.87	960.04	$1^{(*)}$, $2^{(-)}$, $4^{(-)}$	1058.08	76.65	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	50	0.01	2718.06	340.38	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	2549.63	299.87	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	1506.3	175.58	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	1363.06	122.48	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	1000	50	0.001	5350.41	879.86	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	2885.41	344.11	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	4042.55	602.53	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	1692.8	144.07	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	50	0.0001	9604.51	2304.1	$2^{(-)}$, $3^{(*)}$, $4^{(-)}$	3178.61	362.85	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	8825.42	1766.56	$1^{(*)}$, $2^{(-)}$, $4^{(-)}$	1990.33	166.1	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$

TABLE 4.2. Statistical results of total offline error for (1+1)-EA and POSDC with large change ($r = 2000$) in the dynamic constraint with $n = 100$

	τ	δ	α	(1+1)-EA-Chebyshev (1)			(1+1)-EA-Chernoff (2)			POSDC-Chebyshev (3)			POSDC-Chernoff (4)		
				Mean	Std	Stat	Mean	Std	Stat	Mean	Std	Stat	Mean	Std	Stat
uncorrelated	100	25	0.01	194216.1	60110.05	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	241785.41	65345.78	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	78069.93	32609.46	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	88737.18	37384.9	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	100	25	0.001	23986.42	8881.09	$2^{(+)}$, $3^{(-)}$, $4^{(+)}$	171414.29	53277.1	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	10708.18	4310.19	$1^{(+)}$, $2^{(+)}$, $4^{(+)}$	68846.44	28173.31	$1^{(-)}$, $2^{(+)}$, $3^{(-)}$
	100	25	0.0001	11791.66	940.95	$2^{(+)}$, $3^{(-)}$, $4^{(+)}$	127231.1	43081.04	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	7042.84	703.78	$1^{(+)}$, $2^{(+)}$, $4^{(+)}$	53018.97	23263.6	$1^{(-)}$, $2^{(+)}$, $3^{(-)}$
	100	50	0.01	54698.92	22210.24	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	68452.56	26463.82	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	27812.86	12155.5	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	34253.19	15504.7	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	100	50	0.001	11937.75	1729.94	$2^{(+)}$, $3^{(-)}$, $4^{(+)}$	39427.9	15783.4	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	5389.04	746.01	$1^{(+)}$, $2^{(+)}$, $4^{(+)}$	20299.47	9146.1	$1^{(-)}$, $2^{(+)}$, $3^{(-)}$
	100	50	0.0001	16018.12	708.14	$2^{(+)}$, $3^{(-)}$, $4^{(-)}$	26578.49	10606.86	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	12099.82	744.85	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	12555.5	5260.28	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	1000	25	0.01	31186.99	11837.28	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	38261.16	13374.1	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	8297.34	6722.01	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	8717.4	5936.82	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	1000	25	0.001	6255.83	1484.76	$2^{(+)}$, $3^{(-)}$, $4^{(*)}$	27808.98	10406.46	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	3056.27	722.78	$1^{(+)}$, $2^{(+)}$, $4^{(+)}$	6965.12	5441.94	$1^{(*)}$, $2^{(+)}$, $3^{(-)}$
	1000	25	0.0001	8821.17	867.76	$2^{(+)}$, $3^{(-)}$, $4^{(-)}$	21192.14	8515.38	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	6960.69	732.36	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	5695.93	3629.55	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	1000	50	0.01	10328.6	4770.92	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	12248.5	5497.26	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	3802.82	2038.14	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	3815.72	1971.7	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	1000	50	0.001	6790.75	695.26	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	7999.81	3194.97	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	4702.37	564.26	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	3269.02	1445.49	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	50	0.0001	13491.01	1338.88	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	6009.88	1670.76	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	12017.16	1152.05	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	2947.78	903.74	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
bounded-strongly-correlated	100	25	0.01	167881.08	53820.95	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	223285.83	55756.46	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	33480.65	17574.33	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	37196.18	19396.88	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	100	25	0.001	14944.95	5383.33	$2^{(+)}$, $3^{(-)}$, $4^{(+)}$	150457.7	43849.41	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	7693.99	2190.97	$1^{(+)}$, $2^{(+)}$, $4^{(+)}$	30299.37	16471.56	$1^{(-)}$, $2^{(+)}$, $3^{(-)}$
	100	25	0.0001	13512.83	979.04	$2^{(+)}$, $3^{(-)}$, $4^{(*)}$	103089.32	35646.04	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	9315.01	855.82	$1^{(+)}$, $2^{(+)}$, $4^{(+)}$	23323.14	12004.02	$1^{(*)}$, $2^{(+)}$, $3^{(-)}$
	100	50	0.01	33300.16	16222.9	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	46131.1	20247.74	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	13927.2	6308.14	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	15227.33	7630.53	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	100	50	0.001	11317.76	603.01	$2^{(+)}$, $3^{(-)}$, $4^{(*)}$	23471.08	10350.43	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	6972.38	296.93	$1^{(+)}$, $2^{(+)}$, $4^{(+)}$	10532.46	4703.77	$1^{(*)}$, $2^{(+)}$, $3^{(-)}$
	100	50	0.0001	20559.69	2643.86	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	16250.62	6380.33	$1^{(+)}$, $3^{(*)}$, $4^{(-)}$	15189.83	2031.18	$1^{(+)}$, $2^{(*)}$, $4^{(-)}$	7800.55	2880.6	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	25	0.01	31342.99	7231.9	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	37971.88	7235.31	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	3620.98	2957.09	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	3848.78	2583.35	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	1000	25	0.001	5549.74	653.84	$2^{(+)}$, $3^{(-)}$, $4^{(-)}$	28451.18	6224.04	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	2801.6	249.48	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	3057.99	1909.44	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	1000	25	0.0001	8683.48	1569.48	$2^{(+)}$, $3^{(-)}$, $4^{(-)}$	22073.7	5474.2	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	6519.92	1138.64	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	2806.87	1770.85	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	50	0.01	9219.66	3358.95	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	11976.75	4111.56	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	2622.07	864.5	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	2522.08	878.58	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	1000	50	0.001	6600.47	949.4	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	6964.72	1934.28	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	4591.29	707.31	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	2352.11	388.64	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	50	0.0001	12973.88	2999.2	$2^{(-)}$, $3^{(*)}$, $4^{(-)}$	5386.82	935.88	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	10054.16	2230.47	$1^{(*)}$, $2^{(-)}$, $4^{(-)}$	2401.25	258.89	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$

TABLE 4.3. Statistical results of total offline error for NSGA-II with changes in the dynamic constraint with $n = 100$

r	τ	δ	α	uncorrelated						bounded-strongly correlated					
				NSGA-II-Chebyshev (5)			NSGA-II-Chernoff (6)			NSGA-II-Chebyshev (5)			NSGA-II-Chernoff (6)		
				Mean	Std	Stat	Mean	Std	Stat	Mean	Std	Stat	Mean	Std	Stat
500	100	25	0.01	1304.15	197.87	$3^{(+)}$, $4^{(+)}$, $6^{(*)}$	1204.79	198.17	$3^{(+)}$, $4^{(+)}$, $5^{(*)}$	1174.85	157.67	$3^{(+)}$, $4^{(+)}$, $6^{(*)}$	1052.23	119.38	$3^{(+)}$, $4^{(+)}$, $5^{(*)}$
500	100	25	0.001	2956.99	407.55	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	1376.74	216.28	$3^{(+)}$, $4^{(*)}$, $5^{(+)}$	2765.38	433.43	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	1230.18	142.87	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	100	25	0.0001	7480.81	786.29	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	1533.44	254.62	$3^{(+)}$, $4^{(*)}$, $5^{(+)}$	6652.13	1396.9	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	1389.54	138.65	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	100	50	0.01	2061.86	292.58	$3^{(+)}$, $4^{(*)}$, $6^{(-)}$	1877.08	285.22	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	1925.18	287.3	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	1722.58	206.48	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	100	50	0.001	5168.12	616.45	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	2227.32	367.2	$3^{(+)}$, $4^{(*)}$, $5^{(+)}$	4746.56	896.3	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	2070.55	241.35	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	100	50	0.0001	12453.22	1194.92	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	2528.92	407.39	$3^{(+)}$, $4^{(*)}$, $5^{(+)}$	9860	2799.71	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	2409.96	291.32	$3^{(+)}$, $4^{(*)}$, $5^{(+)}$
500	1000	25	0.01	1017.99	59.4	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	921.25	85.64	$3^{(+)}$, $4^{(*)}$, $5^{(+)}$	778.04	77.26	$3^{(*)}$, $4^{(*)}$, $6^{(-)}$	697.76	51.32	$3^{(+)}$, $4^{(*)}$, $5^{(+)}$
500	1000	25	0.001	2914.63	138.12	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	1152.17	110.01	$3^{(+)}$, $4^{(*)}$, $5^{(+)}$	2226.24	281.86	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	868.82	56.94	$3^{(+)}$, $4^{(*)}$, $5^{(+)}$
500	1000	25	0.0001	7507.88	831.51	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	1359.33	133.08	$3^{(+)}$, $4^{(*)}$, $5^{(+)}$	5493.25	996.28	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	1026.39	68.94	$3^{(+)}$, $4^{(*)}$, $5^{(+)}$
500	1000	50	0.01	1918.92	89.57	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	1743.6	135.49	$3^{(+)}$, $4^{(*)}$, $5^{(+)}$	1473.65	165.07	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	1327.74	108.96	$3^{(+)}$, $4^{(*)}$, $5^{(+)}$
500	1000	50	0.001	5284.3	382.78	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	2184.23	168.81	$3^{(+)}$, $4^{(*)}$, $5^{(+)}$	3926.06	595.36	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	1662.4	133.97	$3^{(+)}$, $4^{(*)}$, $5^{(+)}$
500	1000	50	0.0001	11357.61	2124.42	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	2578.59	199.67	$3^{(+)}$, $4^{(*)}$, $5^{(+)}$	7545.75	1621.66	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	1960.94	157.63	$3^{(+)}$, $4^{(*)}$, $5^{(+)}$
2000	100	25	0.01	2092.38	223.85	$3^{(+)}$, $4^{(+)}$, $6^{(*)}$	2004.58	226.22	$3^{(+)}$, $4^{(+)}$, $5^{(*)}$	2206.27	183.61	$3^{(+)}$, $4^{(+)}$, $6^{(*)}$	2070.69	211.93	$3^{(+)}$, $4^{(+)}$, $5^{(*)}$
2000	100	25	0.001	3384.4	339.88	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	2141.9	217.66	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	4189.48	361.51	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	2241.98	172.67	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	100	25	0.0001	7239.86	589.75	$3^{(*)}$, $4^{(+)}$, $6^{(-)}$	2277.36	255.08	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	9401.92	1160.91	$3^{(*)}$, $4^{(+)}$, $6^{(-)}$	2477.31	233.2	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	100	50	0.01	2700.53	306.48	$3^{(+)}$, $4^{(+)}$, $6^{(*)}$	2525.42	289.62	$3^{(+)}$, $4^{(+)}$, $5^{(*)}$	3131.95	233.06	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	2815.09	189.53	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	100	50	0.001	5201.32	490.23	$3^{(*)}$, $4^{(+)}$, $6^{(-)}$	2772.83	320.47	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	6767.42	735.85	$3^{(*)}$, $4^{(+)}$, $6^{(-)}$	3239.24	245.79	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	100	50	0.0001	12237.13	666.95	$3^{(*)}$, $4^{(*)}$, $6^{(-)}$	3009.02	345.87	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	14933	2362.3	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	3563.3	252.68	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	1000	25	0.01	1124.81	139.67	$3^{(+)}$, $4^{(+)}$, $6^{(*)}$	1028.51	123.95	$3^{(+)}$, $4^{(+)}$, $5^{(*)}$	1026.94	121.56	$3^{(+)}$, $4^{(+)}$, $6^{(*)}$	920.91	97.31	$3^{(+)}$, $4^{(+)}$, $5^{(*)}$
2000	1000	25	0.001	2697.38	324.11	$3^{(*)}$, $4^{(*)}$, $6^{(-)}$	1195.4	156.77	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	2644.09	364.63	$3^{(*)}$, $4^{(*)}$, $6^{(-)}$	1101.68	106.77	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	1000	25	0.0001	7112.43	708.88	$3^{(*)}$, $4^{(*)}$, $6^{(-)}$	1345.71	175.98	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	6548.32	1173.6	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	1261.31	117.39	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	1000	50	0.01	1861.03	222.37	$3^{(+)}$, $4^{(+)}$, $6^{(*)}$	1662.83	210.78	$3^{(+)}$, $4^{(+)}$, $5^{(*)}$	1792.99	228.23	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	1597.85	173.99	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	1000	50	0.001	4838.51	529.76	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	2004.37	270.85	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	4629.3	743.4	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	1948.82	192.93	$3^{(+)}$, $4^{(*)}$, $5^{(+)}$
2000	1000	50	0.0001	12121.1	1161.23	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	2292.78	312.66	$3^{(+)}$, $4^{(*)}$, $5^{(+)}$	9869.71	2331.91	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	2255.45	227.8	$3^{(+)}$, $4^{(*)}$, $5^{(+)}$

TABLE 4.4. Statistical results of total offline error for (1+1)-EA and POSDC with small change ($r = 500$) in the dynamic constraint with $n = 300$

	τ	δ	α	(1+1)-EA-Chebyshev (1)			(1+1)-EA-Chernoff (2)			POSDC-Chebyshev (3)			POSDC-Chernoff (4)		
				Mean	Std	Stat	Mean	Std	Stat	Mean	Std	Stat	Mean	Std	Stat
uncorrelated	100	25	0.01	18581.07	9984.25	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	17944.57	8778.83	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	4140.92	1408.1	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	3914.67	1481.62	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	100	25	0.001	16413.73	464.72	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	16520.85	6339.29	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	6875.51	574.51	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	3766.96	765.43	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	100	25	0.0001	24417.31	1976.06	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	15251.23	3334.2	$1^{(+)}$, $3^{(*)}$, $4^{(-)}$	16314.06	762.06	$1^{(+)}$, $2^{(*)}$, $4^{(-)}$	3941.81	608.35	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	100	50	0.01	15697.73	2212.33	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	15185.3	2038.13	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	5156.17	585.19	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	4591.74	623.25	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	100	50	0.001	20115.21	1155.87	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	14997.8	850.91	$1^{(+)}$, $3^{(-)}$, $4^{(-)}$	11429.54	600.39	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	5182.14	636.81	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	100	50	0.0001	33725.05	4261.09	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	15280.2	678.95	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	27681.56	2272.83	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	5733.36	668.17	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	25	0.01	6661.99	188.32	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	6377.29	172.48	$1^{(+)}$, $3^{(-)}$, $4^{(-)}$	1786.53	69.3	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	1486.8	68.72	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	25	0.001	10127.13	253.94	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	6722.42	186.16	$1^{(+)}$, $3^{(-)}$, $4^{(-)}$	5257.91	176.88	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	1817.47	86.06	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	25	0.0001	19723.01	179.08	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	7023.34	211.28	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	15317.98	264.52	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	2109.2	100.4	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	50	0.01	8291.98	175.55	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	7713.67	190.77	$1^{(+)}$, $3^{(-)}$, $4^{(-)}$	3395.62	122.26	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	2812.56	123.82	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	50	0.001	14806.86	268.05	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	8341.69	217.15	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	10089.71	269.88	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	3460.09	153.89	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	50	0.0001	30372.13	958.87	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	8884.96	232.74	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	26856.24	593.2	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	4035.44	183.21	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
bounded-strongly-correlated	100	25	0.01	9105.51	1309.27	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	9088.77	1425.28	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	3415.57	197.31	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	3162.13	186.38	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	100	25	0.001	11380.58	1316.8	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	8888.46	775.27	$1^{(+)}$, $3^{(-)}$, $4^{(-)}$	6022.91	536.66	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	3382.79	188.45	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	100	25	0.0001	19061.43	2879.24	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	9057.68	842.81	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	13927.46	1685.59	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	3556.44	228.19	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	100	50	0.01	9953.71	993.56	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	9550.51	950.07	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	4593.32	340.94	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	4134.61	289.7	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	100	50	0.001	15109.23	2058.16	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	9918.4	1064.21	$1^{(+)}$, $3^{(*)}$, $4^{(-)}$	9809.54	1020.47	$1^{(+)}$, $2^{(*)}$, $4^{(-)}$	4629.87	306.52	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	100	50	0.0001	27456.37	5110.01	$2^{(-)}$, $3^{(*)}$, $4^{(-)}$	10354.45	1149.8	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	23494.45	3662.46	$1^{(*)}$, $2^{(-)}$, $4^{(-)}$	5081.65	362.82	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	25	0.01	5117.07	162.86	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	4877.97	172.07	$1^{(+)}$, $3^{(-)}$, $4^{(-)}$	1601.84	82.54	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	1389.06	73.41	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	25	0.001	7913.26	252.95	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	5169.92	192.57	$1^{(+)}$, $3^{(-)}$, $4^{(-)}$	4035.24	175.79	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	1622.06	84.47	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	25	0.0001	15321.55	670.91	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	5377.23	193.5	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	10785.57	456.64	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	1831.34	88.89	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	50	0.01	6423.88	217.69	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	5959.75	200.37	$1^{(+)}$, $3^{(-)}$, $4^{(-)}$	2739.89	120.14	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	2341.05	107.77	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	50	0.001	11594.91	447.39	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	6474.6	232.18	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	7379.63	307.75	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	2809.67	90.82	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	50	0.0001	22841.04	1292.7	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	6933.59	185.13	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	18277.97	839.84	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	3223.7	109.97	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$

TABLE 4.5. Statistical results of total offline error for (1+1)-EA and POSDC with large change ($r = 2000$) in the dynamic constraint with $n = 300$

	τ	δ	α	(1+1)-EA-Chebyshev (1)			(1+1)-EA-Chernoff (2)			POSDC-Chebyshev (3)			POSDC-Chernoff (4)		
				Mean	Std	Stat	Mean	Std	Stat	Mean	Std	Stat	Mean	Std	Stat
uncorrelated	100	25	0.01	560271.23	345563.77	$2^{(*)}$, $3^{(-)}$, $4^{(*)}$	759605.57	424441.77	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	283743.39	189143.82	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	383168.86	244966.34	$1^{(*)}$, $2^{(+)}$, $3^{(*)}$
	100	25	0.001	101751.86	70153.99	$2^{(+)}$, $3^{(-)}$, $4^{(+)}$	510116.38	316938.2	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	21464.36	12483.33	$1^{(+)}$, $2^{(+)}$, $4^{(+)}$	261840.27	170906.28	$1^{(-)}$, $2^{(+)}$, $3^{(-)}$
	100	25	0.0001	34958.73	7367.0	$2^{(+)}$, $3^{(-)}$, $4^{(+)}$	384348.84	255770.64	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	17430.76	3259.24	$1^{(+)}$, $2^{(+)}$, $4^{(+)}$	185253.51	125792.24	$1^{(-)}$, $2^{(+)}$, $3^{(-)}$
	100	50	0.01	192299.16	144008.93	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	237907.3	167063.98	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	64934.29	46662.9	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	92197.95	64936.19	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	100	50	0.001	45140.13	19212.95	$2^{(+)}$, $3^{(-)}$, $4^{(*)}$	166323.72	114993.33	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	13805.19	3197.68	$1^{(+)}$, $2^{(+)}$, $4^{(+)}$	46936.59	33684.79	$1^{(*)}$, $2^{(+)}$, $3^{(-)}$
	100	50	0.0001	38621.91	3270.51	$2^{(+)}$, $3^{(-)}$, $4^{(-)}$	123760.06	84643.77	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	27126.59	3529.1	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	28036.61	17831.62	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	1000	25	0.01	80428.42	52878.25	$2^{(*)}$, $3^{(*)}$, $4^{(*)}$	112137.17	58533.92	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	47459.38	49081.68	$1^{(*)}$, $2^{(+)}$, $4^{(*)}$	61801.11	65959.67	$1^{(*)}$, $2^{(+)}$, $3^{(*)}$
	1000	25	0.001	15881.08	4676.29	$2^{(+)}$, $3^{(-)}$, $4^{(*)}$	70836.92	42347.74	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	8481.39	4062.79	$1^{(+)}$, $2^{(+)}$, $4^{(+)}$	41897.93	43503.99	$1^{(*)}$, $2^{(+)}$, $3^{(-)}$
	1000	25	0.0001	21048.4	1250.42	$2^{(*)}$, $3^{(-)}$, $4^{(*)}$	47358.18	32172.01	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	14923.35	786.15	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	32571.61	35268.68	$1^{(*)}$, $2^{(+)}$, $3^{(*)}$
	1000	50	0.01	23016.41	15768.95	$2^{(*)}$, $3^{(*)}$, $4^{(*)}$	26535.51	17813.49	$1^{(*)}$, $3^{(*)}$, $4^{(*)}$	15098.21	13774.29	$1^{(*)}$, $2^{(*)}$, $4^{(*)}$	18657.46	19141.9	$1^{(*)}$, $2^{(*)}$, $3^{(*)}$
	1000	50	0.001	16961.63	712.31	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	18463.56	9389.82	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	10332.64	717.71	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	11449.34	9852.65	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	1000	50	0.0001	30311.13	3289.95	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	15345.13	5466.87	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	25262.64	2152.27	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	8636.16	5338.26	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
bounded-strongly-correlated	100	25	0.01	372407.51	216081.73	$2^{(+)}$, $3^{(-)}$, $4^{(-)}$	565043.15	251219.05	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	144534.76	101673.2	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	184728.03	126605.76	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	100	25	0.001	46746.27	30250.2	$2^{(+)}$, $3^{(-)}$, $4^{(*)}$	346509.0	196117.89	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	15077.91	7901.68	$1^{(+)}$, $2^{(+)}$, $4^{(+)}$	127509.58	90625.96	$1^{(*)}$, $2^{(+)}$, $3^{(-)}$
	100	25	0.0001	25368.0	2763.5	$2^{(+)}$, $3^{(-)}$, $4^{(*)}$	228463.87	148629.13	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	14285.59	1783.39	$1^{(+)}$, $2^{(+)}$, $4^{(+)}$	91185.75	65756.09	$1^{(*)}$, $2^{(+)}$, $3^{(-)}$
	100	50	0.01	92094.36	72365.55	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	111261.5	83416.38	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	35179.74	25934.19	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	47883.61	36326.23	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	100	50	0.001	27258.79	7998.87	$2^{(+)}$, $3^{(-)}$, $4^{(*)}$	74565.57	55979.2	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	11276.51	529.1	$1^{(+)}$, $2^{(+)}$, $4^{(+)}$	25867.36	18542.72	$1^{(*)}$, $2^{(+)}$, $3^{(-)}$
	100	50	0.0001	33025.14	6885.64	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	56417.76	40222.83	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	22239.25	4292.37	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	16482.04	10268.22	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	1000	25	0.01	51511.15	24711.68	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	76838.41	22691.68	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	8023.7	11260.48	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	9311.3	13853.57	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	1000	25	0.001	9406.32	1274.57	$2^{(+)}$, $3^{(-)}$, $4^{(-)}$	47958.14	20748.24	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	4483.46	901.7	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	6268.5	8700.16	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	1000	25	0.0001	15543.65	2648.28	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	30107.85	16852.52	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	10214.61	1563.53	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	5186.01	6055.61	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	1000	50	0.01	10457.11	5039.56	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	12659.69	7197.77	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	4603.49	3694.8	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	4428.01	3659.33	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	1000	50	0.001	12057.06	1753.38	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	9309.67	3099.5	$1^{(+)}$, $3^{(-)}$, $4^{(-)}$	7159.78	862.85	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	3796.27	2030.46	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	50	0.0001	23092.44	4891.14	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	8497.97	1085.45	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	16529.18	3317.76	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	3825.55	1204.53	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$

TABLE 4.6. Statistical results of total offline error for NSGA-II with changes in the dynamic constraint with $n = 300$

r	τ	δ	α	uncorrelated						bounded-strongly correlated					
				NSGA-II-Chebyshev (5)			NSGA-II-Chernoff (6)			NSGA-II-Chebyshev (5)			NSGA-II-Chernoff (6)		
				Mean	Std	Stat	Mean	Std	Stat	Mean	Std	Stat	Mean	Std	Stat
500	100	25	0.01	1980.24	229.97	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	1708.35	224.33	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	1980.24	229.97	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	1708.35	224.33	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	100	25	0.001	5447.77	316.27	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	2022.42	234.42	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	5447.77	316.27	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	2022.42	234.42	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	100	25	0.0001	15469.4	337.62	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	2300.73	256.74	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	15469.4	337.62	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	2300.73	256.74	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	100	50	0.01	3592.03	282.27	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	3015.42	272.68	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	3592.03	282.27	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	3015.42	272.68	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	100	50	0.001	10252.75	384.6	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	3668.91	306.13	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	10252.75	384.6	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	3668.91	306.13	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	100	50	0.0001	26877.36	587.34	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	4228.33	308.99	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	26877.36	587.34	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	4228.33	308.99	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	1000	25	0.01	1648.96	27.65	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	1351.3	26.57	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	1648.96	27.65	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	1351.3	26.57	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	1000	25	0.001	5108.33	66.15	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	1675.53	31.39	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	5108.33	66.15	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	1675.53	31.39	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	1000	25	0.0001	15230.15	81.73	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	1959.85	38.45	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	15230.15	81.73	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	1959.85	38.45	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	1000	50	0.01	3250.31	48.01	$3^{(+)}$, $4^{(*)}$, $6^{(-)}$	2659.67	41.78	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	3250.31	48.01	$3^{(+)}$, $4^{(*)}$, $6^{(-)}$	2659.67	41.78	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	1000	50	0.001	9946.25	78.45	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	3311	54.97	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	9946.25	78.45	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	3311	54.97	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	1000	50	0.0001	27099.8	141.77	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	3874.02	67.06	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	27099.8	141.77	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	3874.02	67.06	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	100	25	0.01	42656.33	78746.01	$3^{(+)}$, $4^{(+)}$, $6^{(*)}$	35828.26	58468.71	$3^{(+)}$, $4^{(+)}$, $5^{(*)}$	42656.33	78746.01	$3^{(+)}$, $4^{(+)}$, $6^{(*)}$	35828.26	58468.71	$3^{(+)}$, $4^{(+)}$, $5^{(*)}$
2000	100	25	0.001	31712.27	109248.36	$3^{(-)}$, $4^{(+)}$, $6^{(*)}$	35433.57	70777.69	$3^{(-)}$, $4^{(+)}$, $5^{(*)}$	31712.27	109248.36	$3^{(-)}$, $4^{(+)}$, $6^{(*)}$	35433.57	70777.69	$3^{(-)}$, $4^{(+)}$, $5^{(*)}$
2000	100	25	0.0001	16962.29	4568.34	$3^{(+)}$, $4^{(+)}$, $6^{(*)}$	30832.67	48712.89	$3^{(-)}$, $4^{(+)}$, $5^{(*)}$	16962.29	4568.34	$3^{(+)}$, $4^{(+)}$, $6^{(*)}$	30832.67	48712.89	$3^{(-)}$, $4^{(+)}$, $5^{(*)}$
2000	100	50	0.01	32200.35	58380.08	$3^{(+)}$, $4^{(+)}$, $6^{(*)}$	29883.36	62175.41	$3^{(+)}$, $4^{(+)}$, $5^{(*)}$	32200.35	58380.08	$3^{(+)}$, $4^{(+)}$, $6^{(*)}$	29883.36	62175.41	$3^{(+)}$, $4^{(+)}$, $5^{(*)}$
2000	100	50	0.001	23572.49	56055.95	$3^{(-)}$, $4^{(+)}$, $6^{(*)}$	19896.1	31532.43	$3^{(-)}$, $4^{(+)}$, $5^{(*)}$	23572.49	56055.95	$3^{(-)}$, $4^{(+)}$, $6^{(*)}$	19896.1	31532.43	$3^{(-)}$, $4^{(+)}$, $5^{(*)}$
2000	100	50	0.0001	26065.96	2533.4	$3^{(+)}$, $4^{(*)}$, $6^{(-)}$	23699.71	55440.52	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	26065.96	2533.4	$3^{(+)}$, $4^{(*)}$, $6^{(-)}$	23699.71	55440.52	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	1000	25	0.01	2063.64	333.48	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	1757.84	325.86	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	2063.64	333.48	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	1757.84	325.86	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	1000	25	0.001	5511.49	409.15	$3^{(+)}$, $4^{(*)}$, $6^{(-)}$	2119.2	369.5	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	5511.49	409.15	$3^{(+)}$, $4^{(*)}$, $6^{(-)}$	2119.2	369.5	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	1000	25	0.0001	15310.74	342.3	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	2374.45	358.4	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	15310.74	342.3	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	2374.45	358.4	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	1000	50	0.01	3660.19	351.35	$3^{(+)}$, $4^{(*)}$, $6^{(-)}$	3078.84	369.55	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	3660.19	351.35	$3^{(+)}$, $4^{(*)}$, $6^{(-)}$	3078.84	369.55	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	1000	50	0.001	10236.57	426.25	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	3763.4	453.07	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	10236.57	426.25	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	3763.4	453.07	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	1000	50	0.0001	26258.98	1455.03	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	4314.95	465.46	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	26258.98	1455.03	$3^{(*)}$, $4^{(-)}$, $6^{(-)}$	4314.95	465.46	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$

TABLE 4.7. Statistical results of total offline error for (1+1)-EA and POSDC with small change ($r = 500$) in the dynamic constraint with $n = 500$

	τ	δ	α	(1+1)-EA-Chebyshev (1)			(1+1)-EA-Chernoff (2)			POSDC-Chebyshev (3)			POSDC-Chernoff (4)		
				Mean	Std	Stat	Mean	Std	Stat	Mean	Std	Stat	Mean	Std	Stat
uncorrelated	100	25	0.01	25547.53	10661.06	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	25506.57	10289.1	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	5734.87	1158.14	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	5343.61	1316.22	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	100	25	0.001	26585.9	585.36	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	24227.03	6983.74	$1^{(+)}$, $3^{(-)}$, $4^{(-)}$	10245.63	879.35	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	5594.14	921.5	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	100	25	0.0001	38669.07	2134.32	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	23549.88	3380.8	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	23986.66	943.13	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	5914.13	874.4	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	100	50	0.01	24837.67	2688.24	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	23947.29	2085.95	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	7736.68	840.9	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	6893.94	897.03	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	100	50	0.001	32186.81	1413.84	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	24269.69	927.1	$1^{(+)}$, $3^{(-)}$, $4^{(-)}$	16800.31	905.79	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	7743.77	920.97	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	100	50	0.0001	53113.67	4777.78	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	24732.3	542.8	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	41201.03	2070.37	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	8406.07	956.56	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	25	0.01	10679.09	251.38	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	10255.75	283.04	$1^{(+)}$, $3^{(-)}$, $4^{(-)}$	2623.28	59.04	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	2177.02	48.41	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	25	0.001	15353.99	251.59	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	10689.04	257.01	$1^{(+)}$, $3^{(-)}$, $4^{(-)}$	7336.65	87.71	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	2606.9	50.41	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	25	0.0001	29005.42	265.48	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	10929.47	241.09	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	21381.65	180.38	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	2970.18	62.32	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	50	0.01	12879.75	250.15	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	11984.36	284.16	$1^{(+)}$, $3^{(-)}$, $4^{(-)}$	4799.14	59.07	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	3931.04	76.34	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	50	0.001	21846.35	297.37	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	12830.75	280.57	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	14000.45	132.56	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	4774.03	83.49	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	50	0.0001	45527.61	388.1	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	13542.6	221.6	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	38981.91	91.35	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	5504.22	89.54	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
bounded-strongly-correlated	100	25	0.01	13571.56	891.21	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	13283.16	843.81	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	5064.01	181.52	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	4726.24	174.23	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	100	25	0.001	17313.02	1185.71	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	13524.48	873.45	$1^{(+)}$, $3^{(-)}$, $4^{(-)}$	8682.07	385.56	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	5041.81	172.01	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	100	25	0.0001	28001.34	2273.86	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	13793.69	868.91	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	19265.15	1041.29	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	5273.31	198.47	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	100	50	0.01	15304.23	1044.35	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	14620.03	947.35	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	6711.21	279.52	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	6063.73	184.85	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	100	50	0.001	22502.7	1692.3	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	15250.58	1016.16	$1^{(+)}$, $3^{(-)}$, $4^{(-)}$	13818.95	691.04	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	6673.51	216.82	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	100	50	0.0001	40433.65	4094.78	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	15800.27	1069.97	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	32748.25	2541.67	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	7241.5	232.75	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	25	0.01	8006.07	246.05	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	7605.72	186.1	$1^{(+)}$, $3^{(-)}$, $4^{(-)}$	2490.84	160.57	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	2159.69	130.44	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	25	0.001	11670.1	301.42	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	7984.47	221.99	$1^{(+)}$, $3^{(-)}$, $4^{(-)}$	5766.09	165.05	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	2465.36	101.48	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	25	0.0001	21988.98	515.58	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	8236.1	202.32	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	15220.29	375.16	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	2732.15	123.2	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	50	0.01	9743.23	222.54	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	9057.61	197.93	$1^{(+)}$, $3^{(-)}$, $4^{(-)}$	3978.16	102.95	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	3448.46	127.07	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	50	0.001	16674.56	371.01	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	9684.41	241.01	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	10359.91	251.3	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	4044.27	162.05	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	50	0.0001	33358.68	934.81	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	10202.97	191.17	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	25699.41	692.5	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	4547.63	125.05	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$

TABLE 4.8. Statistical results of total offline error for (1+1)-EA and POSDC with large change ($r = 2000$) in the dynamic constraint with $n = 500$

	τ	δ	α	(1+1)-EA-Chebyshev (1)			(1+1)-EA-Chernoff (2)			POSDC-Chebyshev (3)			POSDC-Chernoff (4)		
				Mean	Std	Stat	Mean	Std	Stat	Mean	Std	Stat	Mean	Std	Stat
uncorrelated	100	25	0.01	1061998.7	814370.17	$2^{(*)}$, $3^{(-)}$, $4^{(*)}$	1439549.69	960146.92	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	482702.1	367723.67	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	708278.81	504437.72	$1^{(*)}$, $2^{(+)}$, $3^{(*)}$
	100	25	0.001	227594.01	174438.01	$2^{(+)}$, $3^{(-)}$, $4^{(*)}$	999424.78	746161.39	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	31372.51	16409.05	$1^{(+)}$, $2^{(+)}$, $4^{(+)}$	446334.52	337016.31	$1^{(*)}$, $2^{(+)}$, $3^{(-)}$
	100	25	0.0001	64221.99	14836.09	$2^{(+)}$, $3^{(-)}$, $4^{(*)}$	763947.72	597826.15	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	30408.55	4418.98	$1^{(+)}$, $2^{(+)}$, $4^{(+)}$	294229.84	229487.59	$1^{(*)}$, $2^{(+)}$, $3^{(-)}$
	100	50	0.01	402875.79	340310.54	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	504409.81	423778.54	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	82281.96	66149.17	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	124975.57	103120.73	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	100	50	0.001	93937.35	53223.23	$2^{(+)}$, $3^{(-)}$, $4^{(-)}$	367874.23	306893.74	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	24816.5	4706.67	$1^{(+)}$, $2^{(+)}$, $4^{(+)}$	62714.42	48211.84	$1^{(+)}$, $2^{(+)}$, $3^{(-)}$
	100	50	0.0001	62615.79	3490.3	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	283227.8	228239.95	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	44462.28	3903.98	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	39071.03	25446.22	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	1000	25	0.01	76989.59	74805.42	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	126797.07	94411.04	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	54626.04	84434.18	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	71234.64	110440.83	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	1000	25	0.001	23395.36	6018.96	$2^{(*)}$, $3^{(-)}$, $4^{(+)}$	72923.77	65146.89	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	10635.11	4616.68	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	52595.23	81013.7	$1^{(-)}$, $2^{(+)}$, $3^{(*)}$
	1000	25	0.0001	32881.96	1815.86	$2^{(*)}$, $3^{(-)}$, $4^{(+)}$	48096.12	44477.47	$1^{(*)}$, $3^{(*)}$, $4^{(-)}$	22113.69	1311.48	$1^{(+)}$, $2^{(*)}$, $4^{(*)}$	36297.22	54405.47	$1^{(-)}$, $2^{(+)}$, $3^{(*)}$
	1000	50	0.01	26997.58	17455.53	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	28846.96	21160.39	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	15766.81	18018.12	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	20829.28	27230.52	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	1000	50	0.001	26694.56	1122.91	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	24108.83	11500.66	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	15292.27	1371.71	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	13017.45	13627.49	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	1000	50	0.0001	46986.42	4628.9	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	21719.8	5746.5	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	37893.14	2786.62	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	10420.59	7285.91	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
bounded-strongly-correlated	100	25	0.01	485045.34	402444.67	$2^{(*)}$, $3^{(-)}$, $4^{(*)}$	778706.81	500486.94	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	203496.58	171490.02	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	298887.93	235686.88	$1^{(*)}$, $2^{(+)}$, $3^{(*)}$
	100	25	0.001	83030.62	77897.12	$2^{(+)}$, $3^{(-)}$, $4^{(*)}$	466446.83	383812.6	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	18030.96	9756.2	$1^{(+)}$, $2^{(+)}$, $4^{(+)}$	187692.66	158975.68	$1^{(*)}$, $2^{(+)}$, $3^{(-)}$
	100	25	0.0001	36162.08	4679.22	$2^{(+)}$, $3^{(-)}$, $4^{(*)}$	310419.86	291916.89	$1^{(-)}$, $3^{(-)}$, $4^{(*)}$	19523.27	2061.6	$1^{(+)}$, $2^{(+)}$, $4^{(+)}$	128385.18	112886.57	$1^{(*)}$, $2^{(*)}$, $3^{(-)}$
	100	50	0.01	151657.79	156077.56	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	183507.1	184812.07	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	40269.97	37317.76	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	57552.97	53820.72	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	100	50	0.001	44437.41	22871.14	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	132170.29	133691.43	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	15653.6	427.4	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	29815.46	25541.87	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	100	50	0.0001	43540.61	8960.02	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	110321.77	107290.04	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	29197.66	5143.73	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	19481.1	12899.68	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	25	0.01	36380.73	26920.16	$2^{(+)}$, $3^{(-)}$, $4^{(-)}$	74613.21	34615.49	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	7310.36	14219.58	$1^{(+)}$, $2^{(+)}$, $4^{(+)}$	7872.99	16402.96	$1^{(+)}$, $2^{(+)}$, $3^{(-)}$
	1000	25	0.001	13182.88	1232.53	$2^{(+)}$, $3^{(-)}$, $4^{(-)}$	36059.29	24473.13	$1^{(-)}$, $3^{(-)}$, $4^{(-)}$	5699.88	420.25	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	6600.89	12281.81	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	1000	25	0.0001	22599.57	2864.27	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	21119.79	18450.2	$1^{(+)}$, $3^{(*)}$, $4^{(-)}$	14349.33	1596.37	$1^{(+)}$, $2^{(*)}$, $4^{(-)}$	5555.48	8360.01	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	50	0.01	12056.59	2618.63	$2^{(*)}$, $3^{(-)}$, $4^{(-)}$	12473.88	5012.82	$1^{(*)}$, $3^{(-)}$, $4^{(-)}$	4796.8	2819.08	$1^{(+)}$, $2^{(+)}$, $4^{(*)}$	4705.07	4048.4	$1^{(+)}$, $2^{(+)}$, $3^{(*)}$
	1000	50	0.001	17630.69	1936.44	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	11611.82	1188.0	$1^{(+)}$, $3^{(-)}$, $4^{(-)}$	9926.29	876.92	$1^{(+)}$, $2^{(+)}$, $4^{(-)}$	4629.46	2459.38	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$
	1000	50	0.0001	33637.01	5237.98	$2^{(-)}$, $3^{(-)}$, $4^{(-)}$	11830.94	788.57	$1^{(+)}$, $3^{(+)}$, $4^{(-)}$	23968.87	3393.71	$1^{(+)}$, $2^{(-)}$, $4^{(-)}$	4655.54	949.69	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$

TABLE 4.9. Statistical results of total offline error for NSGA-II with changes in the dynamic constraint with $n = 500$

r	τ	δ	α	uncorrelated						bounded-strongly correlated					
				NSGA-II-Chebyshev (5)			NSGA-II-Chernoff (6)			NSGA-II-Chebyshev (5)			NSGA-II-Chernoff (6)		
				Mean	Std	Stat	Mean	Std	Stat	Mean	Std	Stat	Mean	Std	Stat
500	100	25	0.01	2629.15	250.1	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	2166.88	242.94	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	1775.95	59.71	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	1486.83	78.45	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	100	25	0.001	7337.29	263.17	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	2604.9	220.96	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	5176.67	121.02	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	1795.8	66.35	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	100	25	0.0001	21372.96	357.01	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	2965.5	223.38	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	14307.84	350.68	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	2062.77	58.96	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	100	50	0.01	4785.34	192.57	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	3947.44	256.57	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	3361.73	74.88	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	2754.17	66.84	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	100	50	0.001	13976.83	294.21	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	4782.73	251.83	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	9776.71	200.09	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	3361.35	78.9	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	100	50	0.0001	38919.03	154.93	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	5502.27	251.65	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	24386.51	616.53	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	3907.85	85.84	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	1000	25	0.01	2243.92	21.42	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	1811.45	20	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	1645.48	17.4	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	1336.88	16.54	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	1000	25	0.001	6965.34	25.98	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	2229.64	15.07	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	5048.74	39.22	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	1647.13	23	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	1000	25	0.0001	20952.86	45.38	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	2599.85	20.35	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	14227.63	107.44	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	1920.8	13.55	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	1000	50	0.01	4436.16	18.89	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	3560.31	22.05	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	3226.71	21.02	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	2618.29	20.4	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
500	1000	50	0.001	13592.41	35.66	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	4400.64	28.67	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	9666.55	68.79	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	3237.62	17.83	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	1000	50	0.0001	38605.89	33.45	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	5131.92	24.39	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	24431.89	191.89	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	3779.76	25.23	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	100	25	0.01	1980727.33	9558032.36	$3^{(-)}$, $4^{(-)}$, $6^{(*)}$	481741.2	1561556.42	$3^{(+)}$, $4^{(+)}$, $5^{(*)}$	14274.08	46280.86	$3^{(+)}$, $4^{(+)}$, $6^{(*)}$	9583.78	21515.62	$3^{(+)}$, $4^{(+)}$, $5^{(*)}$
2000	100	25	0.001	228705.83	693557.41	$3^{(*)}$, $4^{(+)}$, $6^{(*)}$	2099557.82	9963314.09	$3^{(-)}$, $4^{(-)}$, $5^{(*)}$	6998.51	4907.24	$3^{(+)}$, $4^{(+)}$, $6^{(*)}$	14613.43	49357.48	$3^{(-)}$, $4^{(+)}$, $5^{(*)}$
2000	100	25	0.0001	151089.88	378427.61	$3^{(-)}$, $4^{(+)}$, $6^{(*)}$	493694.97	2234147.08	$3^{(-)}$, $4^{(-)}$, $5^{(*)}$	14622	1442.27	$3^{(+)}$, $4^{(*)}$, $6^{(-)}$	8367.86	17466.64	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	100	50	0.01	430142.43	1477435.51	$3^{(*)}$, $4^{(-)}$, $6^{(*)}$	413148.59	1015120.2	$3^{(-)}$, $4^{(-)}$, $5^{(*)}$	6524.52	7782.79	$3^{(+)}$, $4^{(+)}$, $6^{(+)}$	8669.58	20999.17	$3^{(+)}$, $4^{(+)}$, $5^{(-)}$
2000	100	50	0.001	567897.01	2911412.7	$3^{(-)}$, $4^{(*)}$, $6^{(*)}$	805338.65	2374576.61	$3^{(-)}$, $4^{(-)}$, $5^{(*)}$	10878.92	2026.34	$3^{(+)}$, $4^{(*)}$, $6^{(-)}$	6630.53	10554.83	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	100	50	0.0001	67159.26	112444.17	$3^{(*)}$, $4^{(*)}$, $6^{(+)}$	2039569.02	8305881.27	$3^{(-)}$, $4^{(*)}$, $5^{(-)}$	24023.68	3337.45	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	16815.6	59035.06	$3^{(+)}$, $4^{(-)}$, $5^{(+)}$
2000	1000	25	0.01	2731.65	380.74	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	2257.47	298.06	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	1802.3	62.01	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	1501.77	56.55	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	1000	25	0.001	7414.54	329.02	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	2702.24	350.82	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	5152.23	131.64	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	1823.2	78.04	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	1000	25	0.0001	21440.57	481.58	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	3069.28	297.1	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	14162.83	506.49	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	2083.27	69.55	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	1000	50	0.01	4882.2	344.2	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	3992.09	319.33	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	3366.21	86.36	$3^{(+)}$, $4^{(+)}$, $6^{(-)}$	2756.02	69.28	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	1000	50	0.001	14109.65	477.06	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	4889.62	389.76	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	9691.41	276.03	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	3359.4	76.44	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$
2000	1000	50	0.0001	38811.05	218.98	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	5606.6	357.17	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$	24261.29	911.19	$3^{(+)}$, $4^{(-)}$, $6^{(-)}$	3914.22	97.42	$3^{(+)}$, $4^{(+)}$, $5^{(+)}$

Chapter 5

Novelty-Driven Binary Particle Swarm Optimisation for Truss Optimisation Problems

In Chapter 4, we showed that evolutionary algorithms could solve theoretical combinatorial optimisation problems such as the dynamic chance-constrained knapsack problem. This chapter focuses on a problem in engineering design known as truss optimisation. We approach one primary aspect of the problem, including/excluding a truss member, as a combinatorial optimisation problem.

Trusses are structural frameworks that consist of bars connecting its nodes. They carry applied external forces on nodes to support structures such as bridges and towers. Topology optimisation of trusses to determine including and excluding a bar can be formulated as a combinatorial and multi-modal problem. It has been observed that there exist multiple distinct topologies with almost equal overall weight in the truss optimisation search space [DG01; ILM17; Li+16b]. Therefore finding multiple equally good truss designs can enable practitioners to choose according to their preferences.

In this chapter, we consider the bilevel optimisation of trusses with a primary focus on the upper level as a combinatorial optimisation problem. We propose two approaches considering the upper level search space in the truss test problems and assume that the size of bars should be selected from a discrete set with respect to practice code constraints.

Truss optimisation problem could be low and high dimensional with respect to the size of upper level. We introduce exact enumeration to rigorously analyse the topology search space and remove randomness for small problems. Exact enumeration iterates over all possible upper level topologies, and we apply the lower level optimisation to every feasible upper level design. We also propose novelty-driven binary particle swarm optimisation for bigger problems to discover new designs at the upper level by maximising novelty. We show through experimental investigation that our approach outperforms the current state-of-the-art methods and it obtains multiple high-quality solutions.

This chapter is based on the work to appear in European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar) 2022 [Ass+22b]. We added more details for experiments and explanations. The rest of this chapter is organised as follows. In the next section, we state the bilevel truss optimisation problem with the focus on upper level and we explain the lower level optimiser. Next, we introduce the exact enumeration and propose the bilevel novelty search framework including the upper level repair operator. We carry out experimental investigations and report on the quality of obtained solutions for different truss test problems. We finish with some concluding remarks and some suggestions for future work.

5.1 Bilevel Truss Optimisation Problem

Recall the definition of truss optimisation problem in Section 2.4. The formulation has been based on the assumption that topology and size (and shape) optimisation are separable problem. However, these are non-separable problems that makes it inappropriate to use the equation mentioned earlier (Eq 2.4

Bilevel optimisation is a more effective approach than the previous two approaches (see Chapter 2.4) to dealing with truss optimisation problems because it can explicitly model the interaction among different aspects of the problem. In the bilevel formulation, the upper level optimisation problem determines the truss configuration, such as topology, where the lower level optimises bars' sizing.

Islam et al. [ILM17] adopted a bilevel representation for the truss optimisation problem, where the weight of the truss was the main optimisation objective for both upper and lower levels. In the upper level, a modified binary Particle Swarm Optimisation (PSO) with niching capability was used to enhance its population diversity while still maintaining some level of exploitation. The niching technique was based on a speciation scheme that applies a niching radius to subdivide the swarm to locate multiple high-quality solutions. A standard continuous PSO was employed for the lower level to supply good sizing solutions to the upper level. However, using standard PSO for such constrained engineering problems can lead to trapping in local optima [HPW04].

We state the bilevel truss optimisation problem as follows where we embed an upper level topology optimisation problem into a lower level size and shape optimisation problem.

$$
\begin{array}{ll}
\text{find} & \vec{x}, \vec{y}, \quad \vec{x} \in \{0,1\}^m \\
\text{optimise} & F(\vec{x}, \vec{y}) \\
\text{subject to} & G_1(\vec{x}),\, G_2(\vec{x}),\, G_3(\vec{x}) \\
\text{where} & G_1(\vec{x}) = \text{True} \iff \text{Necessary nodes are active in truss} \\
 & G_2(\vec{x}) = \text{True} \iff \text{Truss is externally stable} \\
 & G_3(\vec{x}) = \vec{y} \in \operatorname{argmin}\{W(\vec{x}, \vec{y}),\, g_j(\vec{x}, \vec{y}) \leq 0,\, j = 1,\, 2,\, 3\}
\end{array}
$$

Topology variable $\vec{x}$ is a binary variable in the upper level where it shows if a truss bar is active (1) or excluded (0). m shows the length of upper level topology design representation and determines the low or high dimensional of the search space of topology optimisation. For instance, in 25-bar truss m is 8 due to symmetry that the 25 members are divided into 8 groups. For the same problem, we can show the upper bound of topology as the ground structure where all bars are active as $\vec{x} = [11111111]$.

$\vec{y}$ denotes the design variable in the lower level optimisation problem including size and shape with respect to the test problem. The size variables of $\vec{y}$ should be selected from an available size set (S) and shape variables from a continuous range with respect to the layout constraints of the test problems. $F(\vec{x}, \vec{y})$ shows the objective function considered in the upper level such as weight minimisation used in [ILM17].

Solutions in the upper level should satisfy the topology constraints for feasibility. G_1 enforces that the design should have active nodes that support the truss and carry the external load, because they are necessary elements in the design space's predefined boundary conditions. For example for 25-bar truss (depicted in Figure 5.1) nodes 7-10 are support nodes and nodes 1-3 and 6 are carrying external loads. Therefore these eight nodes are necessary nodes in the design space.

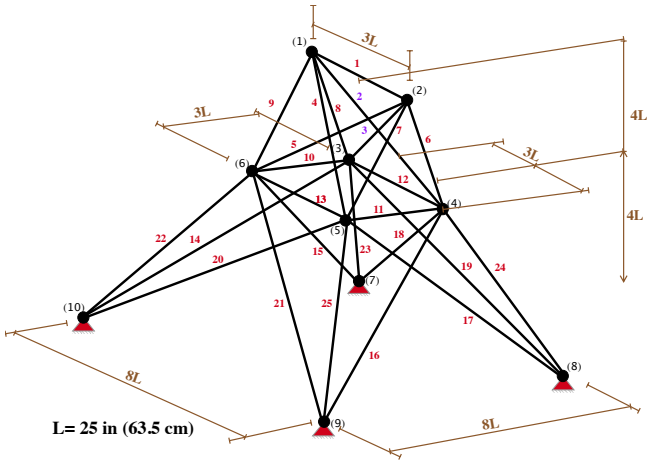

FIGURE 5.1. Ground structure of 25-bar truss.

G_2 states the external stability of a truss. The external stability satisfaction criteria are fully detailed in the following section. Feasible topology solutions should meet G_1 and G_2. In this case, the lower level optimiser aims to find the optimum $\vec{y}$ to minimise the overall weight of truss (W) which is the summation of the weight of all included bars ($\hat{m}$) in the truss.

$$W(\vec{x},\vec{y}) = \rho \sum_{i=1}^{\hat{m}} x_i y_i l_i$$

where ρ and l show the density of the material used in the truss (such as aluminium or steel) and length of a bar with respect to its end points in the design space, respectively. upper level external stability satisfaction is necessary but not sufficient. Therefore, in the lower level, the internal stability should be checked through lower level function evaluation considering that truss stiffness matrix should be positive-definite.

If $(\vec{x},\vec{y})$ meets the internal stable condition, extra constraints should be satisfied. These constraints state that the computed stress in bars ($\sigma_i,\ i \in \{1,2,..,\hat{m}\}$) and displacement of truss nodes ($\delta_k,\ k = \{1,2,..,n\}$) after applying the external loads should not exceed their problem-dependent allowable values $\sigma_i^{max}, i \in \{1,2,..,\hat{m}\}$, and $\delta_k^{max}, k \in \{1,2,..,n\}$, respectively.

The lower level constraints $g_j,\ j = 1,2,3$, used as part of $G_3(\vec{x})$ are therefore defined as follows.

$$\begin{aligned}
&g_1(\vec{x},\vec{y}) = \text{True} \iff \text{Truss is internally stable}\\
&g_2(\vec{x},\vec{y}) \le 0 \iff g_{2,i}(\vec{x},\vec{y}) = \sigma_i - \sigma_i^{max} \le 0 \quad \forall i \in \{1,2,..,\hat{m}\}\\
&g_3(\vec{x},\vec{y}) \le 0 \iff g_{3,k}(\vec{x},\vec{y}) = \delta_k - \delta_k^{max} \le 0 \quad \forall k \in \{1,2,..,n\}
\end{aligned}$$

See Section 2.4 for more details on these constraints.

5.2 Optimisation Methods

In this section we describe the methods used for bilevel optimisation of trusses. We first explain the lower-level optimisation where a domain knowledge-based algorithm computes the optimum design variables for size and shape optimisation. Next, we present the exact enumeration technique applied to small-scale truss problems where

Algorithm 5.1: Lowerlevel optimiser

```
1  while termination criterion is not met do
2      Set the shape and size design variables
3      Set the independent mutation step
4      Initialise the recombinant design
5      Mutating shape variables
6      Mutating size variables
7      Computing the member stress and nodal deflection
8      Calculate the objective function and constraints
9      Resizing the size variables
10     Select the best individuals and update the strategy parameters
```

the upper level topology search space is tractable. Subsequently, we develop the novelty-driven binary PSO to tackle truss problems with bigger topology search space.

5.2.1 Lower Level Optimisation

We use a domain knowledge-based evolutionary optimiser for lower level optimisation [AD16] to determine the optimum layout. The *loweroptimiser* is a variant of Covariance Matrix Adaptation Evolution Strategy algorithm (CMA-ES) that is customised to be problem-specific.

loweroptimiser follows the main principles of evolutionary strategies to evolve the solutions. However, it uses specific operators to adjust solutions with respect to the allowable stress and displacement in the truss. Algorithm 5.1 shows its procedure. It starts with setting the shape and size design variables with length of the nodes and members in the design space. Next, it allocates an independent mutation step for each design variables. Shape variables are mutated in the next step using a truncated normal distribution where the bound constraints are automatically satisfied. After it determined the shape, the size variables are mutated using a stochastic a probabilistic scheme to round the values to the discrete set to avoid biasing the search towards sub-optimal solutions. Next, *loweroptimiser* performs finite element analysis to compute the stress in members and nodal deflection and subsequently calculates the objective function. Moreover, *loweroptimiser* employs a mapping strategy with respect to the response after performing finite element analysis to adjust the sizing of a violating bar by multiplying its current size with a factor that depends on the amount of violation. Another operator is a resizing strategy for producing new individuals near boundary designs of the problem and the problem-dependent constraints. For brevity, we refer the reader to [AAD20; AD16] for detailed explanations on different components of the *loweroptimiser*.

5.2.2 Exact Enumeration

We apply exact enumeration to the truss problems where the upper level dimension of the search space dimension is low ($m \leq 12$). Exact enumeration enables us to enumerate over all possible combinations of binary strings in the search space in the upper level, where each represents a topology design. Therefore we can remove the randomness for these problems from the upper level and investigate its search space rigorously. Algorithm 5.2 shows the pseudocode of our exact enumeration. This algorithm takes the upper level dimension m as the input and iterates over all possible binary string combinations of m-bits. If a binary string satisfies the upper

Algorithm 5.2: Exact Enumeration

```
for i = 1; i ≤ 2^m; i = i + 1 do
    compute x_i                                  ▷ generate the bit string
    if x_i is feasible then
        y_i = loweroptimiser(x_i)
    Store W(x_i, y_i)
```

level's feasibility criteria G_1 and G_2, it will be sent to the lower level. *loweroptimiser* aims to find the optimum vector for size (and shape), and we store its overall weight.

5.2.3 Novelty-Driven Bilevel Truss Optimisation

In this section we introduce the components of the proposed bilevel method. Next, we combine these components and explain the framework of the algorithm.

Binary PSO

Recall PSO from Section 3.3.3 which is typically used as a continuous optimisation algorithm. Therefore, to use it for binary search spaces, we need to employ a transfer function, such as Sigmoid transfer function, to map a continuous search space into a binary search space [KE95]. To determine if i^{th} element of a binary string of particle (z_i) should be 0 or 1 we have:

$$\int(v_t^i) = \frac{1}{1 + e^{-v_t^i}}$$

where v_t^i refers to the i^{th} element of the velocity vector at iteration t and $\int(.)$ denotes the transform function and to determine the i^{th} element of z_i. We have

$$z_t^i = \begin{cases} 1 & \text{if } rand() \geq \int(v_t^i) \\ 0 & \text{otherwise,} \end{cases}$$

where $rand()$ is a uniformly drawn random number from $[0, 1]$.

In this thesis we employ a time-varying transfer function [ILM17] to balance between exploration and exploitation. Velocities of all particles are updated according to the following velocity update equation to determine the probabilities for flipping the position vector elements (i).

$$z_i^t = \begin{cases} 1 & \text{if } rand() \geq TV(v_i^t, \varphi) \\ 0 & \text{otherwise,} \end{cases}$$

where TV is given as [ILM17],

$$TV(v_i^t \varphi) = \frac{1}{1 + e^{-\frac{v_i^t}{\varphi}}}$$

φ is the control parameter to balance exploration and exploitation in the course of optimisation where it linearly decreases from 5.0 to 1.0 [ILM17].

Novelty-driven Binary PSO

Novelty-Driven PSO (NdPSO) is a variant of PSO employing novelty search to drive particles toward novel solutions that are different from previously encountered ones [FLP16]. The main idea is to explore the search space by ignoring objective-based fitness functions and reward novel individuals. NdPSO uses the score of novelty to evaluate the performance of particles. For this purpose, it maintains an archive of past visited solutions to avoid repeatedly cycling through the same series of behaviours. NdPSO evaluates the novelty of particles by computing the average distance of a behaviour to its k-nearest neighbours in the archive as follows:

$$\text{nov}(x) = \frac{1}{k}\sum_{i=0}^{k} dist(x, \mu_i).$$

where μ_i is the i^{th} nearest neighbour of x and $dist$ is the Hamming distance metric. Novelty score ensures that individuals in less dense areas with respect to the archive get higher novelty scores.

NdPSO employs core principles of PSO and mainly replaces the objective function with novelty evaluation. Note that personal best and global best value in NdPSO show a dynamic behaviour. For more details on NdPSO, we refer the reader to [FLP16]. We use NdPSO in the upper level of truss optimisation to discover novel topology designs.

Repair Mechanism in the Upper Level

Topology designs in the upper level are feasible if they meet $G_1(\vec{x})$ and $G_2(\vec{x})$. The following conditions should be satisfied for $G_2(\vec{x})$ [AD16]:

- The degree of freedom (DoF) in a truss should be non-positive [DG01].
- The summation of the number of members and restrain forces on a node must be equal or greater than the truss dimension.
- The summation of the number of members and restraint forces on a non-carrying node must be greater than the truss dimension.

As stated in [DG01], necessary node constraints are more important than the DoF constraint. To deal with infeasible topologies, we use the (1+1)-EA (See Algorithm 3.5) with the following comparator to repair solutions:

$$\begin{aligned} x \succeq y := &(\alpha(x) \leq \alpha(y)) \vee \\ &(\alpha(x) = \alpha(y) \wedge \beta(x) \leq \beta(y)) \vee \\ &(\alpha(x) = \alpha(y) \wedge \beta(x) = \beta(y) \wedge \theta(x) \leq \theta(y)) \end{aligned}$$

where α is the violation degree of active necessary nodes, β is the violation degree of truss DoF and θ is the violation degree of second and third criteria in external stability. (1+1)-EA as described in Algorithm 3.5 is a simple evolutionary algorithm where it produces an offspring by mutation and the offspring replaces the parent if it is determined to be at least as good or better according to the fitness function mentioned above.

Algorithm 5.3: Bilevel Novelty-Driven Binary PSO Framework for Truss Optimisation

```
Randomly generate the initial population of Binary PSO
Repair the initial population into feasible topologies
Set the velocity of particles in population
Evaluate the novelty score for each particle
y⃗ = loweroptimiser(x⃗)
Store W(x⃗, y⃗)
Update p_t and p_g
Update the archive
repeat
    for i=1 to population size do
        Update position of particle
        Update velocity of particle
        Repair the particle into feasible upper level solution
        Evaluate novelty score of the particle
        y⃗_i = loweroptimiser(x⃗_i)
        Store W(x⃗_i, y⃗_i)
        Update p_t^i and p_g according to novelty score
        Update the archive
until termination criterion is met
```

Bilevel Novelty-Driven Binary PSO Framework

Our proposed approach works as follows (see Algorithm 5.3). Initially, the binary PSO generates a random population of binary strings. This string represents the inclusion and exclusion of truss bar members with respect to the ground structure. Next, the repair mechanism performs on the population to ensure the feasibility of particles with respect to the upper level feasibility constraints. The particles' velocities are drawn randomly from $[-\upsilon, \upsilon]$. Then, the novelty score is computed for particles with respect to the archive. Because all particles are feasible after using repair mechanism, *loweroptimiser* computes the corresponding optimal size (and shape) for the upper level topology. With this, we update the archive with the current population. Next, the position and velocity of particles are updated, and the above process repeats till the termination criterion is met.

5.3 Experimental Investigations

In this study, we use multiple truss test problems with discrete sizing from the literature [AD16; DLU19; LHL09]. We investigate them in ascending order of length of topology design variable. We use the best reported weights from state-of-art for our comparison.

We split the problems into small and large instances representing low and high dimension of the topology search space. We apply exact enumeration to small problems where their topology search space is tractable ($m \leq 12$). To show its outcome, we set the ground structure of the problem as an upper bound reference. We sort the other designs with respect to their Hamming distance with this reference design (denoted by d_H). We report on the quality of solutions using the median of best solutions obtained

in 30 independent runs in the lower level. For large instances, we apply the proposed bilevel novelty search PSO and report on the quality of top best-found solutions.

We investigate the obtained designs and identification of redundant bars and nodes in the design space. We report on the three best found solutions with different topologies represented by designs (a), (b) and (c) in the corresponding tables.

To setup our algorithms, we use the following parameters. For lower level optimisation, we use the parameter settings in [AD16; AAD20]. For the upper level optimisation, the swarm consisted of 30 particles, $\upsilon = 6$, $c_1 = c_2 = 1.0$, ω linearly decreases from 0.9 to 0.4 [ILM17], and the maximum number of iterations is set to 300. We use $k = 3$ nearest neighbours to calculate novelty at the upper level.

The implemented framework is available on `https://bit.ly/3pFNdFb`. This framework is in Matlab and included all instances used in this chapter namely, 25-bar, 10-bar, 52-bar, 15-bar, 72-bar, 47-bar, 224-bar, 200-bar and 68-bar instances. Each instance can be simulated by simply running its own Matlab file. The framework also includes the other components of the proposed algorithms including exact enumeration and binary novelty PSO.

5.3.1 25-Bar Truss

25-bar truss, as aforementioned is a spatial truss for size and topology optimisation [DLU19]. This truss problem is a symmetric truss where 25 truss bars are grouped into eight groups of bars. Therefore, there are 256 possible topologies at the upper level.

This problem splits into two cases with respect to different load cases. The truss undergoes a single load for the first case and multiple external loads for the second case. The sizing of bars for the first and second cases should be selected from different sets of available cross-sections [DLU19]

Because the topology search space is tractable, we apply the exact enumeration to this test problem. Figure 5.2 shows the outcome of the exact enumeration of the first case study of this test problem. We can see the designs shown with respect to their topology and weight. Note that the squares are coloured according to the weight of the obtained design. This figure shows only the first 50 designs with respect to the hamming distance (d_H) with the upper bound reference as the ground structure. These are feasible designs, and the rest of the search space is infeasible. We can see that the high-quality solutions are in the upper bound vicinity where $d_H \leq 2$.

Figure 5.3 depicts 55 feasible designs for the second case of the 25-bar truss problem. We can see that the ground structure topology results in a reasonable light truss similar to the first case. However, the high-quality designs are located in the region where $d_H \leq 3$.

Tables 5.1 and 5.2 shows our findings for case 1 and 2, respectively compared with the state of art solutions. We can see that designs (b) and (c) both have identified two solutions with $d_H = 1$. However, design (a) represents the best-found solution where it combines the identified redundant members in designs (b) and (c), and it also adjusts the sizes of bars to meet truss constraints leading to a lightweight solution. For the second case, we can see that all top designs identify G_5 as redundant and similar to the first case, the best design (a) incorporates both identified redundant bar groups of designs (b) and (c) and adjust the size variables to reach the lightest truss.

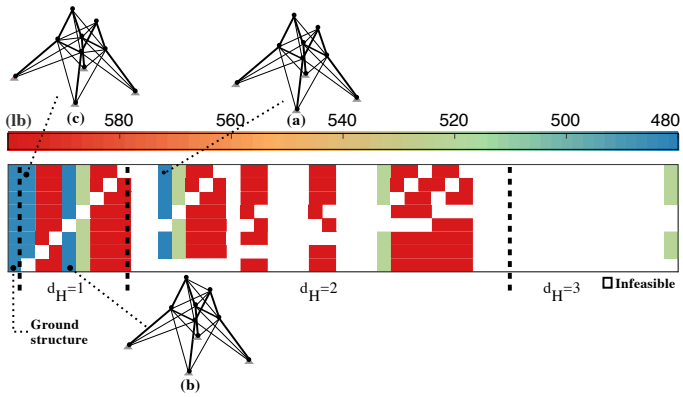

FIGURE 5.2. Exact enumeration on 25-bar truss case 1 (right side truncated). d_H denotes the hamming distance with the upper bound reference. Note that empty area denotes the infeasible region of the search space.

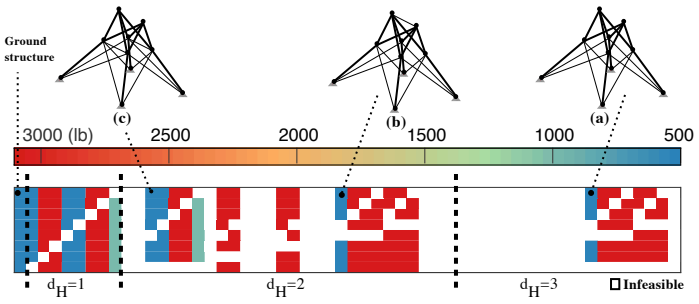

FIGURE 5.3. Exact enumeration on 25-bar truss case 2 (right side truncated). d_H denotes the hamming distance with the upper bound reference. Note that empty area denotes the infeasible region of the search space.

5.3.2 10-Bar Truss

10-bar truss [LHL09] is a well-known size and topology optimisation problem whose ground structure is depicted in Figure 5.4. It undergoes a single load, and the sizing of bars should be selected from a discrete set where there are 10 bars in the topology design, which results in 1024 possible upper level topologies. Therefore we can apply the exact enumeration.

Figure 5.5 shows the outcome of exact enumeration, and the designs are ordered according to their d_H. We only show the first 320 sorted designs because the rest of the topologies are infeasible. This figure also shows the top obtained designs for this test problem. We can also see many feasible heavy trusses are close to the designs reported, which shows the complexity of the upper level search space that an algorithm can trap in a local optimum in the topology search space.

We can see feasible designs are located where $d_H \leq 4$ and the best-found design's d_H is 4. Table 5.3 shows our findings. We can see that designs (b) and (c) with $d_H = 2$ both identify two bars as redundant and incorporate all six nodes in their designs compared to the design (a), which is the best-found design that identifies four bars (A_2, $A_5 - A_6$ and A_{10}) as redundant and eliminates node 6. We can see that both designs (b) and (c) incorporate two extra members (which are potentially redundant) to include node 6 in their designs with different topology, and the difference lies in including and excluding members A_2 and $A_1 0$. Note that node 6 is not necessary because it

TABLE 5.1. Comparison of optimised designs for 25-bar truss case 1.

	[RK92]	[Ho-+16]	[Che16]	This Study		
				(a)	(b)	(c)
G_1	0.1	0.1	0.1	-	0.1	-
G_2	1.8	0.3	0.3	0.3	0.3	0.5
G_3	2.3	3.4	3.4	3.4	3.4	3.4
G_4	0.2	0.1	0.1	-	-	0.1
G_5	0.1	2.1	2.1	2.1	2.1	1.9
G_6	0.8	1	1	1	1	0.9
G_7	1.8	0.5	0.5	0.5	0.5	0.5
G_8	3	3.4	3.4	3.4	3.4	3.4
Best weight (lb)	546.01	484.85	484.85	482.6	483.35	484.3

TABLE 5.2. Comparison of optimised designs for 25-bar truss case 2.

	[Lee+05]	[LHL09]	This Study		
			(a)	(b)	(c)
G_1	0.307	0.111	-	0.111	-
G_2	1.99	2.13	2.13	2.13	2.13
G_3	3.13	2.88	2.88	2.88	2.88
G_4	0.111	0.111	-	-	-
G_5	0.141	0.111	-	-	0.111
G_6	0.766	0.766	0.766	0.766	0.766
G_7	1.62	1.62	1.62	1.62	1.62
G_8	2.62	2.62	2.62	2.62	2.62
Best weight (lb)	556.43	551.14	546.97	547.81	548.64

neither carries load nor supports the truss. So it can potentially be eliminated from the design space. We can also see that other state of art methods, including [Fen+14] and [Kha+20] also identified the same topology as the optimal topology. However, our approach can obtain a solution with a lower weight due to the efficient lower level optimiser.

5.3.3 52-Bar Truss

This truss problem is a size and topology optimisation problem where 52-bar truss are grouped into 12 bar groups resulting in 4096 possible designs. Therefore we can apply the exact enumeration to this truss test problem. The truss undergoes three load cases, and the sizing of bars should be selected from a discrete set [WC95]. The ground structure of the 52 bar truss problem has been shown in Fig. 5.6 (I).

Figure 5.7 shows the outcome of exact enumeration where the figure only illustrates the first 250 designs sorted according to their d_H. Out of all sorted combinations, 1900 designs are feasible, and the rest of the search space is infeasible. We also observed that feasible designs are located where $d_H \leq 6$.

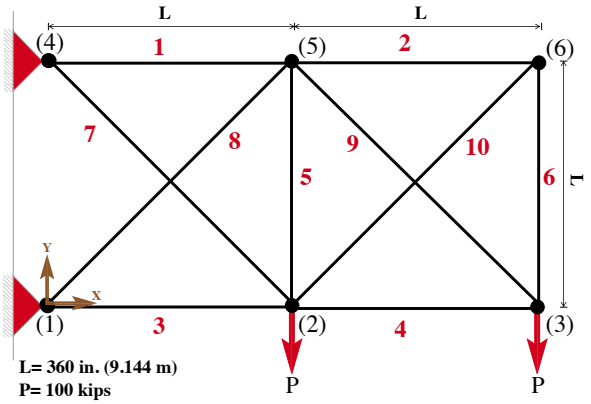

FIGURE 5.4. Ground structure of 10-bar truss.

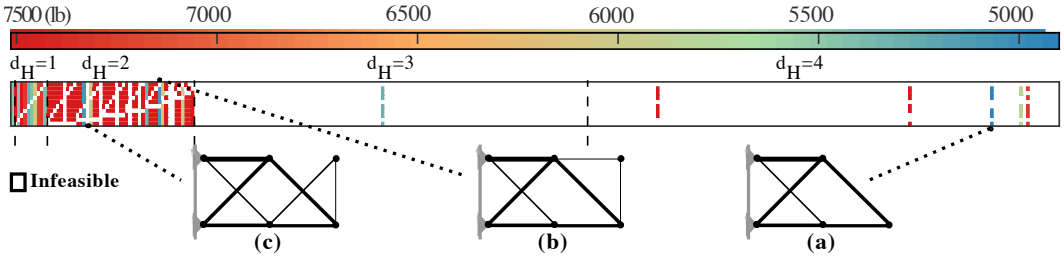

FIGURE 5.5. Exact enumeration on 10-bar truss. d_H denotes the hamming distance with the upper bound reference.

Table 5.4 shows our findings, and we can see that the top three designs for this problem identify one to three groups of bars as redundant, respectively. Design (a) depicted in Figure 5.6 (II) shows the best-found design that eliminates all redundant bars identified in designs (b) and (c) and removes G_9 resulting in nine redundant bars (3 bar groups).

5.3.4 15-Bar Truss

This truss test problem is a size and topology truss optimisation problem, which is a non-symmetric truss composed of 15 bars and the truss undergoes three load cases, and the sizing should be selected from a discrete set [Zha+05]. Figure 5.8 (I) shows the ground structure of this truss.

Table 5.5 shows our findings by the proposed bilevel novelty search compared with other methods. All found top topologies identify five different bars as redundant, resulting in different topologies with different weights. We can see that designs (b) and (c) find the same weight, and they are symmetric around the vertical axis with respect to the topology and size of bars. Both designs remove 6 bars from the design space, and symmetrically they eliminate nodes 2 and 4 from the design space, respectively. Design (d) eliminates five bars and node two from the design space.

Figure 5.8 shows these three designs. Design (a) as the best-found design depicted, design (a) eliminates five bars in the design space and provides a lighter solution than other methods.

5.3.5 72-Bar Truss

72-bar truss represents a four-storey structure for size and topology optimisation where it is a symmetric truss composed of 72 bars grouped into 16 groups of bars. The truss

TABLE 5.3. Comparison of optimised designs for 10-bar truss.

	[LHL09]	[Ho-+16]	[Che16]	[Fen+14]	[Kha+20]	This Study		
						(a)	(b)	(c)
A_1	30	33.5	33.5	29.5	30	30	30	30
A_2	1.62	1.62	1.62	-	-	-	1.62	-
A_3	22.9	22.9	22.9	23.6	22	22	22	22
A_4	13.5	14.2	14.2	16.8	13.9	13.5	14.2	14.2
A_5	1.62	1.62	1.62	-	-	-	-	-
A_6	1.62	1.62	1.62	-	-	-	1.62	1.62
A_7	7.97	7.97	7.97	6.1	7.22	7.22	7.22	7.22
A_8	26.5	22.9	22.9	21	22	22	22	22
A_9	22	22	22	22.8	22	22	22	22
A_{10}	1.8	1.62	1.62	-	-	-	-	1.62
Best weight (lb)	5531.98	5490.74	5490.74	5056.88	4980.10	4965.7	5107.5	5131.7

TABLE 5.4. Comparison of optimised designs for 52-bar truss.

	[WC95]	[Lee+05]	[LHL09]	[KT09]	[Ho-+16]	[Che16]	This Study		
							(a)	(b)	(c)
G_1	4658.055	4658.055	4658.055	4658.055	4658.055	4658.055	5141.925	4658.055	4658.055
G_2	1161.288	1161.288	1161.288	1161.288	1161.288	1161.288	939.998	1045.159	1008.385
G_3	645.160	506.451	363.225	494.193	494.193	494.193	-	-	-
G_4	3303.219	3303.219	3303.219	3303.219	3303.219	3303.219	3703.218	3703.218	3703.218
G_5	1045.159	940.000	940.000	1008.385	939.998	940.000	792.256	939.998	1008.385
G_6	494.193	494.193	494.193	285.161	494.193	494.193	-	-	198.064
G_7	2477.414	2290.318	2238.705	2290.318	2283.705	2238.705	2477.414	2477.414	2341.931
G_8	1045.159	1008.385	1008.385	1008.385	1008.385	1008.385	939.998	1008.385	939.998
G_9	285.161	363.225	388.386	388.386	494.193	494.193	-	198.064	285.161
G_{10}	1696.771	1535.481	1283.868	1283.868	1283.868	1283.868	1535.481	1535.481	1535.481
G_{11}	1045.159	1045.159	1161.288	1161.288	1161.288	1161.288	1045.159	1008.385	1008.385
G_{12}	641.289	506.451	792.256	506.451	494.193	494.193	363.225	494.193	388.386
Best weight (kg)	1970.140	1906.760	1905.500	1904.830	1902.610	1902.610	1862.000	1880.300	1869.7

undergoes two load cases, and the sizing of bars should be selected from a set [WC95].

Table 5.6 shows our findings by the proposed bilevel novelty search compared with other methods. Designs (b) and (c) identify five bar groups as redundant, including four common groups where the ultimate design eliminates 16 bars from the design space. Design (a) as the best-found design, combines the identified redundant bars in designs (b) and (c) and removes 20 bars in total from the design space and achieves a lighter solution.

5.3.6 47-Bar Truss

This truss problem represents a transmission tower with symmetric truss bars grouped into 27 groups [HE01], and it considers size and shape optimisation in the lower level. See [HE01] for structural constraints and technical information to simulate the problem. Table 5.7 shows our findings in comparison with state of the art. We can see that all obtained designs identify the $G_{21} - G_{22}$ and G_{27} as redundant.

We can see that our method obtained three different designs with respect to the topology. Design (a) incorporates 23 topology bars, and design (b) and (c) both include 21 groups of bars, and the main difference is including bar groups of 23 and

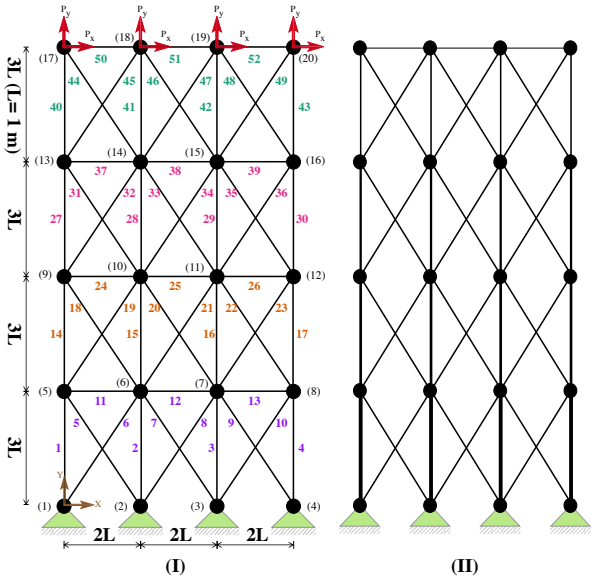

FIGURE 5.6. Ground structure of 52-bar truss (I) and the best found design (II)

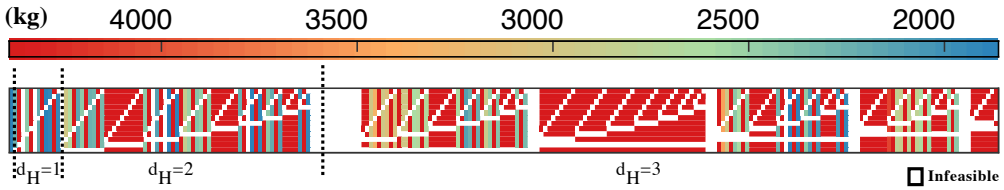

FIGURE 5.7. Exact enumeration on 52-bar truss (right side truncated).

26, respectively. We can also observe that even though designs (b) and (c) identify two other bar groups as redundant compared to design (a), they result in heavier structure than the best-found solution.

5.3.7 200-Bar Truss

This size and topology optimisation problem includes 200-bars grouped into 29 bar groups [KT09]. The truss undergoes three loading conditions and is subject to no displacement constraint.See [KT09] for details on simulation. Table 5.8 shows our findings and compared with reported weights in state of the art. The best-reported weight is 25156.5 lb, but this is an infeasible design because it violates the stress constraint by about 8%. We can see that Design (a) as the best-found solution where similar to design (c) both include 24 group bars but with different topologies; however, Design (b) incorporates all group members as 29.

Moreover, the obtained design (b) topology is as same as the ground structure where all bars are included in the topology. This is similar to our observation of the 25-bar truss problem, where the ground structure results in a reasonable lightweight truss. We can also observe that design (c) recognises 4 group bars as redundant. Still, it doesn't necessarily result in a light truss compared to design (b), including all possible connections of the bars.

TABLE 5.5. Comparison of optimised designs for 15-bar truss.

	[Zha+05]	[Che16]	This Study			
			(a)	(b), (c)		(d)
A_1	308.6	113.2	113.2	-	113.2	-
A_2	174.9	113.2	113.2	-	113.2	-
A_3	338.2	113.2	-	113.2	-	113.2
A_4	143.2	113.2	-	113.2	-	113.2
A_5	736.7	736.7	736.7	736.7	736.7	736.7
A_6	185.9	113.2	-	113.2	113.2	113.2
A_7	265.9	113.2	143.2	143.2	143.2	143.2
A_8	507.6	736.7	736.7	736.7	736.7	736.7
A_9	143.2	113.2	113.2	-	113.2	-
A_{10}	507.6	113.2	-	113.2	-	-
A_{11}	279.1	113.2	113.2	145.9	145.9	145.9
A_{12}	174.9	113.2	113.2	-	-	-
A_{13}	297.1	113.2	-	-	-	113.2
A_{14}	235.9	334.3	334.3	334.3	334.3	334.3
A_{15}	265.9	334.3	334.3	334.3	334.3	334.3
Best weight (kg)	142.12	105.74	89.899	90.223		91.874

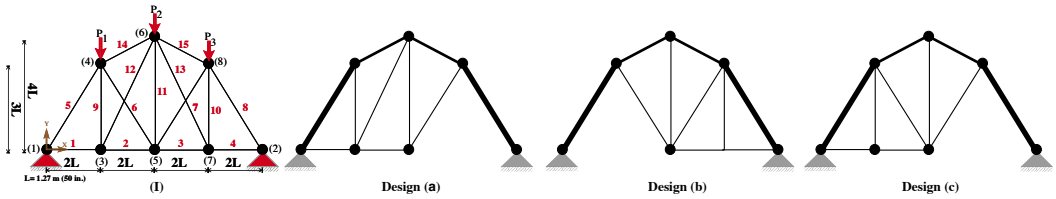

FIGURE 5.8. The ground structure of 15 bar truss (I) and top three designs obtained (a-c)

5.3.8 224-Bar Truss

This problem represents a pyramid truss where the truss bars are grouped into 32 groups and considers size, shape and topology optimisation subject to complex design specifications [HE01]. Table 5.9 lists obtained designs by our method compared with state of the art. We can see that all methods have identified the same 16 bar groups as redundant out of 32 possible bar groups. Design (a) outperforms other designs, and design (b) is as alike as the optimum design obtained in [AD16]. Design (c) is different from the other two designs by including G_9 with a slight increase in the weight. Therefore if the designer prefers to have G_9 in the final design, it can result in a reasonable lightweight truss compared to other obtained solutions.

5.3.9 68-Bar Truss

This truss problem is a size, shape and topology optimisation problem with multi-load with 68 non-symmetric topology design variables [AD16]. The optimum design should be feasible considering the structural reactions subject to 8 different loading

TABLE 5.6. Comparison of optimised designs for 72-bar truss.

	[WC95]	[Lee+05]	[Che16]	This Study		
				(a)	(b)	(c)
G_1	1.5	1.9	1.9	2	2	1.9
G_2	0.7	0.5	0.5	0.5	0.5	0.5
G_3	0.1	0.1	0.1	-	-	-
G_4	0.1	0.1	0.1	-	-	-
G_5	1.3	1.4	1.4	1.4	1.3	1.3
G_6	0.5	0.6	0.5	0.6	0.5	0.5
G_7	0.2	0.1	0.1	-	-	-
G_8	0.1	0.1	0.1	-	-	0.1
G_9	0.5	0.6	0.5	0.5	0.5	0.5
G_{10}	0.5	0.5	0.5	0.5	0.5	0.6
G_{11}	0.1	0.1	0.1	-	-	-
G_{12}	0.2	0.1	0.1	-	0.1	-
G_{13}	0.2	0.2	0.2	0.2	0.2	0.2
G_{14}	0.5	0.5	0.6	0.5	0.6	0.5
G_{15}	0.5	0.4	0.4	0.5	0.5	0.5
G_{16}	0.7	0.6	0.6	0.6	0.6	0.7
Best Weight (lb)	400.66	387.94	385.54	368.16	369.15	370.15

conditions. Table 5.10 shows our findings in comparison with state of the art. This table only lists the bars if they are present in one of the listed designs; otherwise, it has been identified as redundant by all designs.

The best solution reported is by reported by [AD16] is the best-found solution. Our method has obtained three different designs where design (a) is similar to the best of the state-of-art. Designs (b) and (c) include 37 and 39 bars, respectively, out of 68 possible bars. The main difference in design (a) with the other designs is that the former identifies bars A_{48}, $A_{51} - A_{52}$, $A_{54} - A_{59}$ and $A_{64} - A_{66}$ redundant.

5.4 Conclusions

Weight minimisation of trusses is a multi-modal problem where locating distinct optimal designs can enable practitioners to choose the ultimate design according to their preferences such as in real practice, the size of truss bars should be selected from a set of available sizes according to design codes. In this chapter we considered bilevel optimisation for truss problems where the topology optimisation is a combinatorial optimisation problem. We proposed two approaches to tackle small and large scale problems. In our experiments, we analysed the search space of smaller problems without randomness in the upper level using exact enumeration. We developed a novelty-driven binary PSO to tackle the other problems where the topology search space is bigger. We also observed that we can find multiple distinct high-quality solutions with respect to the topology – moreover, we have found new best solutions for 8 out of 9 test problems. Bilevel optimisation problems nest an optimisation problem

into another where it increases the computational expense. This is the main drawback of this study. We also setup our algorithms with standard parameters. For future studies, it could be interesting (1) to investigate automated tuning of the algorithm, (2) to study this approach for large-scale trusses and (3) to improve it considering the computational expense of the problem. This chapter serves as a link between benchmark problems and practical, real-world problems. By designing computer programs and examining a well-known engineering design problem, we gained insights into how to approach problems from different angles avoiding randomness in methods. Additionally, incorporating domain knowledge could help us find better solutions to these problems. Note that In this study, a binary PSO was employed. Additional studies can be conducted to compare the performance of the current method against traditional binary methods commonly utilised in combinatorial optimization. This will help to gain a better understanding of the strengths and limitations of the method and its applicability to related problems.

TABLE 5.7. Comparison of optimised designs for 47-bar truss.

	[HE02]	[PB18]	[DLU18]	[AD16]	This Study		
					(a)	(b)	(c)
G_1	2.6	3.1	2.7	3	2.6	2.4	3.4
G_2	2.4	1.1	2.5	0.3	0.4	0.4	0.3
G_3	0.8	3	0.7	2.6	2.6	2.3	3.1
G_4	-	1.1	0.1	1.5	0.4	0.3	0.6
G_5	1.1	2.8	0.9	2.8	2.5	2.2	2.8
G_6	1.3	1.1	1.1	0.6	0.4	0.5	0.4
G_7	1.7	2.6	1.8	2.3	1.8	1.7	2.4
G_8	0.6	0.9	0.7	1.1	0.5	0.4	0.8
G_9	1	2.7	0.9	2.5	2.2	2.3	2.6
G_{10}	1.4	0.7	1.3	0.6	0.5	0.3	0.7
G_{11}	0.5	2.6	0.3	2.3	2.1	1.6	1.7
G_{12}	1.1	0.8	1.1	1.4	1.6	1.9	1.9
G_{13}	1	0.8	1	0.7	1.7	0.5	0.6
G_{14}	1	1.7	0.9	1.6	1.5	1.8	2
G_{15}	0.8	1	0.8	0.9	0.9	0.9	1.2
G_{16}	-	1	0.1	0.8	0.4	0.7	1.2
G_{17}	2.7	0.3	2.7	0.2	0.5	1	1.1
G_{18}	1	1	0.8	0.9	1	1	1
G_{19}	-	1.3	0.1	1.2	1.3	1.4	1.3
G_{20}	2.9	0.9	3	1.1	1.4	0.7	1
G_{21}	0.9	0.9	0.9	1.2	-	-	-
G_{22}	-	1.2	0.1	-	-	-	-
G_{23}	3.1	0.1	3.2	-	1	0.6	-
G_{24}	1.1	0.1	1	-	0.1	-	-
G_{25}	-	0.1	0.1	-	0.4	-	-
G_{26}	3.2	0.1	3.3	-	-	-	0.2
G_{27}	1	0.1	1.1	-	-	-	-
Best weight (lb)	1885.070	1871.700	1836.462	1727.624	1724.947	1726.044	1727.624

TABLE 5.8. Comparison of optimised designs for 200-bar truss where † denotes the reported solution is infeasible.

	[Che16]	[Ho-+16]	[KT09]	[DLU19]	This Study		
					(a)	(b)	(c)
G_1	0.1	0.1	0.1033	0.1	-	0.1	-
G_2	0.954	0.954	0.9184	0.954	1.081	0.954	1.081
G_3	0.1	0.347	0.1202	0.347	0.347	0.347	0.44
G_4	0.1	0.1	0.1009	0.1	0.1	0.1	0.347
G_5	2.142	2.142	1.8664	2.142	2.142	2.142	2.142
G_6	0.347	0.347	0.2826	0.347	0.347	0.347	0.44
G_7	0.1	0.1	0.1	0.1	-	0.1	-
G_8	3.131	3.131	2.9683	3.131	3.131	3.131	3.131
G_9	0.1	0.347	0.1	0.1	0.347	0.1	0.347
G_{10}	4.805	4.805	3.9456	4.805	4.805	4.805	4.805
G_{11}	0.44	0.539	0.3742	0.44	0.44	0.44	0.44
G_{12}	0.347	0.347	0.4501	0.347	-	0.347	-
G_{13}	5.952	5.952	4.96029	5.952	5.952	5.952	5.952
G_{14}	0.347	0.1	1.0738	0.1	0.1	0.1	0.539
G_{15}	6.572	6.572	5.9785	6.572	6.572	6.572	6.572
G_{16}	0.954	0.954	0.78629	0.954	0.539	0.954	0.954
G_{17}	0.347	0.44	0.73743	0.1	-	0.347	0.1
G_{18}	8.525	8.525	7.3809	8.525	8.525	8.525	8.525
G_{19}	0.1	0.1	0.6674	0.539	0.954	0.1	0.347
G_{20}	9.3	9.3	8.3	9.3	9.3	9.3	9.3
G_{21}	1.081	0.954	1.19672	0.954	0.954	0.954	0.954
G_{22}	0.347	1.081	1	0.1	-	1.081	-
G_{23}	13.33	13.33	10.8262	13.33	13.33	13.33	13.33
G_{24}	0.954	0.539	0.1	0.1	0.44	0.539	0.539
G_{25}	13.33	14.29	11.6976	13.33	13.33	13.33	13.33
G_{26}	1.764	2.142	1.388	0.954	1.174	2.142	1.081
G_{27}	3.813	3.813	4.9523	5.952	5.952	3.813	5.952
G_{28}	8.525	8.525	8.8	10.85	10.85	8.525	10.85
G_{29}	17.17	17.17	14.6645	14.29	14.29	17.17	14.29
Best weight (lb)	27163.59	27858.50	25156.50†	27282.54	27144.0	27575.0	27744.0

TABLE 5.9. Comparison of optimised designs for 224-bar truss.

	[HE01]	[HE02]	[AD16]	This Study		
				(a)	(b)	(c)
G_1	17.29	-	-	-	-	-
G_2	27.74	11.9	-	-	-	-
G_3	27.74	-	-	-	-	-
G_4	4.32	-	-	-	-	-
G_5	14.52	-	36.0	36.0	36.0	36.0
G_6	14.39	-	-	-	-	-
G_7	17.29	8.4	-	-	-	-
G_8	14.39	-	6.90	6.90	6.90	6.90
G_9	3.16	0.8	-	-	-	0.8
G_{10}	9.55	0.49	-	-	-	-
G_{11}	6.9	-	-	-	-	-
G_{12}	4.32	-	-	-	-	-
G_{13}	10.97	1.07	-	-	-	-
G_{14}	6.9	-	-	-	-	-
G_{15}	17.29	0.67	-	-	-	-
G_{16}	17.29	8.4	27.74	27.74	27.74	20.45
G_{17}	27.74	0.8	17.29	20.45	17.29	36.0
G_{18}	9.55	0.8	-	-	-	-
G_{19}	6.9	0.8	-	-	-	-
G_{20}	-	1.07	-	-	-	-
G_{21}	-	1.7	4.32	4.32	4.32	4.32
G_{22}	17.29	1.07	23.74	20.45	23.74	10.97
G_{23}	-	-	3.19	4.32	3.19	6.90
G_{24}	14.39	-	-	-	-	-
G_{25}	-	8.4	3.19	5.15	3.19	6.90
G_{26}	27.74	0.49	17.29	17.29	17.29	27.74
G_{27}	-	-	2.79	2.79	2.79	2.79
G_{28}	-	-	6.90	6.90	6.90	6.90
G_{29}	-	-	27.74	27.74	27.74	20.45
G_{30}	0	-	1.61	1.61	1.61	2.15
G_{31}	10.97	1.07	17.2	20.45	17.29	36.0
G_{32}	14.39	2.23	4.12	3.19	4.12	2.06
Best weight (lb)	5547.500	4587.290	3079.446	3063.866	3079.446	3102.079

TABLE 5.10. Comparison of optimised designs for 68-bar truss.

	[PB18]	[AD16]	This Study				[PB18]	[AD16]	This Study		
			(a)	(b)	(c)				(a)	(b)	(c)
A_1	2.142	3.131	3.131	0.22	0.22	A_{30}	3.131	-	-	-	-
A_2	3.813	1.333	1.333	0.954	0.954	A_{31}	-	-	-	-	0.141
A_3	-	3.131	3.131	3.813	3.813	A_{32}	0.111	2.142	2.142	0.44	-
A_5	-	0.539	0.539	-	-	A_{33}	-	1.333	1.333	0.111	0.141
A_6	1.333	1.333	1.333	0.174	0.539	A_{34}	-	0.954	0.954	0.111	-
A_7	0.954	1.081	1.081	0.954	0.954	A_{35}	-	1.333	1.333	-	0.954
A_8	1.081	2.142	2.142	1.764	1.764	A_{37}	-	0.347	0.347	0.22	0.22
A_9	1.174	0.347	0.347	0.44	0.27	A_{38}	-	1.333	1.333	-	-
A_{10}	-	0.347	0.347	-	-	A_{39}	-	0.539	0.539	0.174	0.27
A_{11}	-	3.131	3.131	3.131	3.131	A_{40}	0.539	1.174	1.174	0.347	0.174
A_{13}	2.697	0.44	0.44	0.22	0.22	A_{41}	2.697	1.333	1.333	0.141	0.954
A_{14}	-	-	-	0.141	-	A_{42}	-	1.081	1.081	0.287	0.27
A_{15}	0.954	1.081	1.081	0.27	0.44	A_{43}	-	-	-	0.22	0.111
A_{16}	3.131	1.488	1.488	1.488	1.488	A_{44}	2.8	-	-	-	-
A_{17}	-	0.27	0.27	0.27	-	A_{47}	-	0.111	0.111	0.111	0.44
A_{18}	-	0.44	0.44	0.539	0.111	A_{48}	-	-	-	-	0.27
A_{19}	-	2.8	2.8	3.131	2.8	A_{51}	-	-	-	0.22	0.111
A_{20}	0.44	-	-	-	-	A_{52}	-	-	-	-	0.27
A_{21}	0.27	0.44	0.44	0.954	0.111	A_{54}	0.111	-	-	-	-
A_{22}	-	0.111	0.111	-	-	A_{55}	-	-	-	-	0.347
A_{23}	1.174	0.27	0.27	0.287	0.111	A_{56}	2.8	-	-	0.954	0.347
A_{24}	-	2.142	2.142	2.142	2.697	A_{57}	-	-	-	-	0.111
A_{25}	-	-	-	-	0.111	A_{58}	-	-	-	0.111	-
A_{26}	-	0.111	0.111	0.111	0.111	A_{59}	3.565	-	-	-	-
A_{27}	2.142	2.697	2.697	0.111	0.954	A_{64}	-	-	-	0.111	-
A_{28}	-	-	0	-	-	A_{65}	-	-	-	2.697	1.764
A_{29}	-	0.111	0.111	-	0.111	A_{66}	-	-	-	1.764	2.142
Best weight (lb)	1385.800	1166.062	1166.062	1167.528	1169.039						

Chapter 6

Modelling and Optimisation of Run-of-Mine Stockpile Recovery

In this chapter, we focus on a more practical application of a combinatorial optimisation problem. We look at a real-world scheduling problem in mining industry where minor changes in the schedule can result in financial penalties for mine operators.

There are two stages to mining: upstream and downstream. Mining upstream involves the extraction of materials from mines, such as ore (valuable minerals) or waste. Haul trucks transport mineral materials extracted from mines either directly to the crusher in downstream for processing or to the Run-of-Mine (ROM) stockpiles. ROM stockpiles are essential components in the mining value chain can be used as temporary storage to balance inflow and outflow and provide an opportunity for blending material. Blending refers to mixing different stacked material in different stockpiles with different quality to assemble a delivery for the customer.

Customers request minerals with a particular percentage of quality. Stockpile schedulers are responsible for selecting a blend of material from the ROM stockyard to provide the delivery. The blend can include high-quality material can be mixed with low-quality material to meet the target quality grade. The stockpile scheduler should plan the deliveries with respect to the end-user requirements such as chemical concentration and particle size distribution limits. We refer to this operation where the scheduler plans how the stockpiles should be reclaimed for deliveries as *stockpile recovery*.

The task of stockyard management is to schedule the reclaiming operations, i.e. to determine the sequence of reclamation operations for end-user requests. In scheduling, it is essential to achieve the quality grade of deliveries and to perform stockpile recovery operations considering the operation costs efficiently. Stockpiles usually have been considered as a whole in the previous studies represented by a weighted average quality. However, in the context of mining stockpiles, Lu and Myo [LM11; LM10] study stockpile recovery optimisation considering that stockpiles are not a whole and can be divided into cuts with different qualities. Cuts refer to a portion of stacked material in the stockpile that represent different grades of mineral resources.

Their model considers a bucket wheel reclaimer which is a machine mainly used in mine terminals for both stacking and reclaiming. They aimed to minimise the movement by a bucket wheel reclaimer machine. For this purpose, they calculated the movement by Euclidean distance, and they applied mixed integer programming for a small scale of the problem considering two requests and two stockpiles in a single period. However, their approach and model is limited: (1) the calculation of reclaimer machine movement cost in practice is more complicated than a simple Euclidean distance; (2) the stockyard in practice is bigger and the resolution of cuts can determine the size and complexity of the stockpile recovery problem; (3) in planning for deliveries, it is essential to schedule deliveries for a longer period to foresee the future; (4)

reclaimer machines also can be other types of machines that can reclaim in different directions; and (5) more technical restrictions exist than a weighted average quality to meet the specified requirements in practice.

In this chapter we model the stockpile recovery problem as a combinatorial optimisation problem considering technical restrictions in real-world settings and we investigate multiple scenarios and experiments. We consider that four stockpiles are stacked in a row in a stockyard, and a single reclaimer machine performs reclamation operations, such as front-end loaders or bucket wheel reclaimers. Reclamation operations require that machines move in the stockyard and reclaim cuts from the stockyard. The cuts in the stockpiles can vary in size (such as increment of 1000-5000 tonnes of materials as per our industry partner) depending on the mineral resources. The reclaiming operation takes time as machinery moves from one position to another and spends time reclaiming, and thus costs are incurred for this operation.

This chapter is based on the work published in 36th Annual ACM Symposium on Applied Computing [Ass+21]. The rest of this chapter is organised as follows. We define the problem as a combinatorial optimisation problem. Next, we introduce the lexicographic objective function and its components to address technical restrictions and stockpile management objectives, we introduce a lexicographic objective function for optimisation considering the importance of target grade qualities and operation costs where the former has a higher priority. Afterwards, we present the scenarios of the problem for investigation. We use the information provided by our industrial partner to simulate the stockyard, where they created cuts and stockpiles using real depositing GPS data of trucks in practice. We define multiple scenarios and experiments to represent a stockyard in practice with different complexities of technical restrictions in the problem. Following, we describe the optimisation algorithms including greedy algorithm and ant colony optimisation and our methodology to deal with the problem. Because we are dealing with a real-world problem, we model the problem and use methodologies align with this point that we can tackle various objective functions, non-linear constraints and further technical restrictions for this study and other extensions. We set up experiments and report on the behaviour and quality of obtained solutions and algorithms for different scenarios. We benchmark against a rule of thumb heuristic called the Pilgrim Step Reclaiming Heuristic. We observe that the randomisation of greedy algorithms can help obtain better solutions in the scenarios, and that ant colony optimisation can outperform all algorithms. Finally, we finish with some concluding remarks and some suggestions for future work.

6.1 Stockpile Recovery Problem Statement

We define the stockpile recovery optimisation problem as a combinatorial optimisation problem. Figure 6.1 depicts a schematic of the stockyard used in this chapter. There is a single reclaimer operating on a row with access to four stockpiles. Due to the information received from our industry partner, only four stockpiles were able to be utilised in the simulation to imitate the single-direction reclaiming method using a single machine.

Each stockpile is comprised of four benches where each bench has 10 cuts. Therefore in total there are 160 cuts that should be sequenced to provide deliveries. We can identify a cut in the stockyard by its position in the stockyard. We show a cut by ($id_{\text{stockpile}}$-id_{bench}-id_{cut}) where for example, cut (1-1-1) shows the first cut in first bench of first stockpile. Note that reclaimers before start reclaiming a stockpile can access the stockpile from the entry cuts.

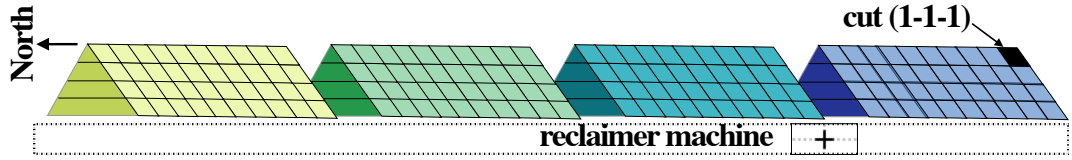

FIGURE 6.1. Schematic of the stockyard. Cut (1-1-1) is the entry cut for stockpile recovery where it is the first cut on stockpile 1, top bench and first cut from South-to-North direction.

The stockyard's structure includes information on multiple stockpiles and information about cuts in each stockpile. We model a stockyard as a directed graph $G = (\mathcal{C}, \mathcal{E})$ without cycles, where $\mathcal{C} = \{c_1, c_2, \ldots, c_n\}$ is the set of cuts containing n available cuts in the stockyard. Each cut c_j has multiple properties including the percentage of valuable and contaminant elements in the cut denoted by $m_{j,k}$, where $k = 1, 2, \ldots, K$ shows the set of chemical elements. Recall from Section 2.6 that chemical constraints are essential to be considered in deliveries to avoid financial penalty fees. Another property denotes how much material is available in the cut by measuring the tonnage of c_j denoted by Γ_j.

$\mathcal{E}$ is the set of edges connecting cuts and shows the immediate predecessors where cut i must be reclaimed before j when the reclaimer direction is d and we denote a reclamation operation by (i, j, d). Precedence constraints determine the validity of a solution. If it fails for a part of the solution, the solution is impossible to be processed. For example we can not reclaim cut (1-2-1) before we reclaim the cut above it (which is (1-1-1)). To perform a reclamation operation, a reclaimer machine can move between two cuts i and j and perform reclamation operation on c_j.

A candidate solution x is a sub-sequence as a permutation of cuts in $\mathcal{C}$, where it represents the order of reclamation to fulfill the optimisation problem's objectives. The total time required to perform a reclamation job from c_i to c_j with direction d is given by $\mathcal{T}_{i,j,d}$. $\mathcal{T}_d$ shows the reclamation time matrix, which is a full square matrix for all cuts in each direction.

As mentioned before, it is essential to preserve the quality of deliveries. One objective is the average target quality which aims to preserve the quality of deliveries where the average of chemical contaminants in a delivery should be in a predefined range as follows for a set of chemical elements (K)

$$\hat{m}_{x_k} = \frac{1}{|x|} \sum_{j=1}^{|x|} m_{k,j}$$

where $\hat{m}_{x_k}$ shows the average of chemical contaminants for cuts in in x with respect to penalty chemical element k.

$$\underline{m_k} \leq \hat{m}_{x_k} \leq \overline{m_k} \quad \forall k \in K$$

Because the chemical properties of cuts have different magnitudes with their corresponding lower and upper bounds, we evaluate the degree of violation for target quality using the bracket-operator penalty method $\langle \hat{m}_{x_k} \rangle$ [Deb01], which is calculated as follows:

$$\langle \hat{m}_{x_k} \rangle = \begin{cases} \frac{|\hat{m}_{x_k} - \overline{m_k}|}{|\overline{m_k}|} & \text{if } \hat{m}_{x_k} > \overline{m_k} \\ \frac{|\hat{m}_{x_k} - \underline{m_k}|}{|\underline{m_k}|} & \text{if } \hat{m}_{x_k} < \underline{m_k} \\ 0 & \underline{m_k} \leq \hat{m}_{x_k} \leq \overline{m_k} \end{cases}$$

To calculate the violation for average target quality in x, we use:

$$v_1(x) = \sum_{k=1}^{K} \langle \hat{m}_{x_k} \rangle$$

Another objective in stockyard recovery is the window target quality. Following a certain number of reclamation operations, reclaimed material is packaged. This objective ensures that when the stockyard scheduler prepares the packages for reclaimed material, the quality of each package should meet another pre-defined window target quality.

$$v_2(x) = \sum_{j=4}^{|x|} \left\langle \frac{(m_{j,k} + m_{j-1,k} + m_{j-2,k})}{3} \right\rangle$$

where j here denotes the position of a cut in a delivery where we desire to look at the window target quality after three cuts are already reclaimed for a delivery.

We would like to reduce the operation costs as as well as financial penalties. A desirable objective is to have a schedule with ability to reclaim more material in a shorter time which it will result in reducing operation costs. For reclamation operation (i, j, d) we have:

$$u(x) = \sum_{(i,j,d) \in x} \frac{T_{i,j,d}}{\Gamma_j}$$

The numerator indicates the corresponding element from reclamation time matrix to perform a job and the denominator is the tonnage of cut j available in stockyard information. We refer to this objective function as the utility.

6.1.1 Objective Function

As a stockyard manager, the primary objective is to avoid paying financial penalty fees with respect to the quality objectives of deliveries. If it is possible to provide a zero-violation solution where all quality objectives are satisfied. The next objective is to reduce the operation costs. Note that this reflects the preferences of our industry partner. Other companies might aim for different goals. To achieve these objectives, we define the objective function for minimisation in lexicographic order:

$$f(x) = (v_1(x), v_2(x), u(x))$$

where order in the objective function matters. To compare tuples of two solutions (x and y):

$$\begin{aligned} & f(x) \leq f(y) \\ \text{iff} \quad & v_1(x) \leq v_1(y) \vee \\ & (v_1(x) = v_1(y) \wedge v_2(x) \leq v_2(y)) \vee \\ & (v_1(x) = v_1(y) \wedge v_2(x) = v_2(y) \wedge u(x) \leq u(y)) \end{aligned}$$

For example, between quality constraints, $v_1(x)$ is more important than $v_2(x)$.

6.1.2 Scenarios Of The Problem

We define three scenarios for investigation of the optimisation problem with different complexities. As we proceed, the problem conditions becomes more similar to the real-world issue in the stockyard management. For each scenario, we consider four

experiments for a more detailed investigation. We show an experiment in a scenario by SC X-Y where X and Y refer to the scenario and experiment numbers, respectively.

In the first scenario, we simulate a stockpile recovery problem where we aim to reclaim the whole stockyard. This scenario helps us investigate the quality of algorithms where technical restrictions are loose, and algorithms can demonstrate their behavior in exploring the stockyard to obtain a reclaiming sequence. We increase the size of stockyard in experiments, where SC1-1, SC1-2, SC1-3 and SC1-4 show that the stockyard contains 1, 2, 3, and 4 stockpiles, respectively, and the termination criterion in solution construction is reclaiming the whole stockyard. In this scenario, we also neglect $v_2(x)$ to make it easier to tackle. For its objective function, we have

$$f_1(x) = (v_1(x), u(x))$$

In the second scenario, we simulate a problem considering technical restrictions in the stockyard. We aim to optimise the reclaiming sequence considering deliveries. We define a sequence for a delivery as x' where $x' \in x$. Similar to the first scenario, we neglect the window target quality constraint. We define different experiments, considering the number of deliveries, where SC2-1, SC2-2, SC2-3, SC2-4 show experiments for second scenario where we plan for one, two, three and four deliveries respectively. Moreover, we consider the stockyard as being initially full, and thus include all stockpiles for reclamation. The termination criterion is constructing a solution where it plans for all deliveries in the experiment. We define the objective function as:

$$f_2(x) = \sum_{x' \in x} \big(v_1(x'), u(x')\big)$$

In the third scenario, we make the problem more similar to real-world problem by technical restrictions: all criteria are as same as the second scenario, but we consider the window target quality too. Our objective function is

$$f_3(x) = \sum_{x' \in x} \big(v_1(x'), v_2(x'), u(x')\big)$$

6.2 Optimisation Methods

In this section, we describe the algorithms applied to solve the scheduling problem of ROM stockpile recovery using a single reclaimer. The search space of the problem is constrained by precedence constraints that impose how the cuts can be reclaimed with respect to physical requirements. To deal with these constraints, we develop methodologies that can construct a valid reclaiming sequence step by step. In this manner, we can easily tackle the precedence constraints, and it can always lead to a valid solution with respect to these constraints. We construct the solution step by step considering available cuts in the neighbourhood to choose as a successor for a solution. We investigate deterministic and randomised greedy algorithms, ant colony optimisation, and its variant with local search operators. We also describe a heuristic representing a rule-of-thumb in stockyard management, namely the pilgrim step reclaiming heuristic (PSRH). We compare the iterative optimisation methods with PSRH to evaluate the performance and quality of our algorithms.

6.2.1 Greedy Algorithm And Randomisation

Recall the application of greedy algorithms in combinatorial optimisation from Section 3.2.3. Greedy algorithms are straightforward and fast in constructing a solution for an optimisation problem, and they are easy to implement. Greedy algorithms can be either deterministic or randomised.

Deterministic greedy algorithm (DGA) constructs a schedule with considering the highest benefit at the time it chooses a cut to put into the schedule. Algorithm 6.1 shows the procedure for DGA which starts with an $S = \emptyset$ representing the empty schedule, and at each time step (t), it adds a successor component (cut) to x. For the stockpile recovery problem, the entry cut pre-defined by the problem is the primary successor at the initial stage of the algorithm. Next, DGA collects the available cuts in the initial cut neighbourhood as $\mathcal{N}_t$ considering the precedence constraints. DGA evaluates how the quality of a schedule changes if it includes each candidate in $\mathcal{N}_t$. Therefore, DGA computes the objective function $f(c_j)$ for performing reclamation from the initial cut to the next cut for each cut in $c_j \in \mathcal{N}_t$. DGA chooses the successor cut with the highest greediness as follows and the algorithm terminates when all the deliveries are scheduled.

$$c^* = \underset{c_j \in \mathcal{N}_t}{\operatorname{argmin}} f(c_j)$$

.

DGA may choose the best cuts in the stockyard early in the planning. This act of greediness can lead to reclaiming all good material in the stockyard early and not having enough good material to blend for later deliveries. Therefore, the algorithm is getting trapped in local optima to plan deliveries, because the individual deliveries are put together in a strict sequential order. To tackle this drawback, other studies (e.g. [Gao+18]) suggest that controlling the greediness can lead to better solutions and avoid the local optima.

Randomised Greedy Algorithm (RGA) is a simple randomised version of DGA, which gives it the potential to find better solutions. Some may argue that the use of the term "Randomised Greedy Algorithm" (RGA) is inappropriate because a greedy algorithm, by definition, cannot be truly random.

RGA has a greedy control parameter $\lambda \geq 0$ where the selection of the successor cut (c^*) occurs according to the probability distribution given by

$$p(c_j|\mathcal{N}_t) = \frac{\eta(c_j)^\lambda}{\sum_{c_l \in \mathcal{N}_t} \eta(c_l)^\lambda}$$

where $\eta(c_j)$ denotes the greedy function to make sure that a better candidate has a higher chance to be selected in the neighbourhood as follows.

$$\eta(c_j) = \frac{1}{f'(c_j)}$$

where $f'(c_j)$ is a mapped real number of $f(c_j)$ required for selection procedure among cuts in the neighbourhood. We classify all cuts in the neighbourhood in three sets $(\mathcal{S})$. Set 1 contain zero-violation solutions, set 2 include solutions with $v_1(c_j) = 0 \wedge v_2(c_j) \neq 0$ and other solutions are put in set 3. For each set separately, we normalise the objective functions component wise to $(0, 1)$ by a linear mapping and we denote it by $\mathcal{N}(c_j)$:

$$\mathcal{N}(c_j) = \mathcal{N}\left(u(c_j)\right) + \mathcal{N}\left(v_2(c_j)\right) + \mathcal{N}\left(v_1(c_j)\right)$$

Algorithm 6.1: Deterministic Greedy Algorithm (DGA)

$x := \emptyset$
repeat
  Choose a successor cut $c^* = \text{argmin}_{c_j \in \mathcal{N}_t} f(c_j)$
  $x := x \cup c^*$
until x *is complete*
return x

Algorithm 6.2: Randomised Greedy Algorithm (RGA)

$x := \emptyset$
repeat
  Choose a successor cut (c^*) according to probability
  $p(c_j|\mathcal{N}_t) = \dfrac{\eta(c_j)^\lambda}{\sum_{c_l \in \mathcal{N}_t} \eta(c_l)^\lambda}$
  $x := x \cup c^*$
until x *is complete*
return x

to calculate $f'(c_j)$, we use:

$$f'(c_j) = \begin{cases} \mathcal{N}(c_j) + 1 & \text{if } c_j \in \mathcal{S}_1 \\ \mathcal{N}(c_j) + 1 + 10 & \text{if } c_j \in \mathcal{S}_2 \\ \mathcal{N}(c_j) + 1 + 10 + 100 & \text{Otherwise.} \end{cases}$$

Note that the addition of 1 ensures that $p(c_j|\mathcal{N}_t) > 0$ and that the addends of 10 and 100 helps to partially preserve the order of objective function.

Our probabilistic selection represents a roulette wheel selection where the chance of selecting a candidate in $\mathcal{N}_t$ is proportionate to its fitness. λ determines how greedy RGA acts for the selection procedure: as $\lambda \to \infty$, RGA approaches the behaviour of DGA. Algorithm 6.2 shows the procedure for RGA. RGA starts from an empty set as well as DGA and use the primary component of the entry cut. When it constructs a schedule at each step, it doesn't opt for the best option available, but it choose the successor component (cut) probabilistically. Therefore, RGA computes the greedy function $\eta(c_j)$ for performing reclamation from the initial cut to the next cut for each cut in $c_j \in \mathcal{N}_t$. Next, RGA chooses the successor cut probabilistically in a fitness proportionate manner. The algorithm terminates when all the deliveries are scheduled.

6.2.2 Max-Min Ant System (MMAS)

In this section, we investigate the MMAS variant of ACO on solving the stockpile recovery problem using single reclaimer. We introduced ACO in Section 3.9. ACO in our study can be viewed as an iterative and adaptive RGA with more control parameters. For our scheduling problems, ants are agents performing a random walk to construct a valid scheduling plan. On their return route to the colony, they deposit pheromone and other ants can sense it and identify a good route instead of a random walk. Other ants follow a route, depositing more pheromone, and thus reinforce the route. In addition, pheromone evaporates gradually to reduce the attraction capability

of untraveled edges. Pheromone is used to represent the quality of different paths or solutions to a problem. The pheromone value is typically updated as the ants explore the search space, with higher pheromone values being associated with better solutions. As the ants explore the search space, some of the pheromone that they deposit on the path they take will gradually evaporate over time. This process is known as pheromone evaporation and is used to reduce the attractiveness of paths that have not been followed in a while. This helps to balance exploration and exploitation and to ensure that the ants do not get stuck in local optima.

For this study, we use a variant of ACO, namely Max-Min Ant System (MMAS) [SH00]. In MMAS, only selected ants deposits pheromone to reinforce its solution where in this Chapter, the best ant at each generation does it. The pheromone range is limited to avoid becoming very big or very small. This feature can prevent getting trapped in local optimal. Algorithm 6.3 shows the procedure of MMAS for stockpile recovery problem. First, MMAS initialises the pheromone matrix for the initial generation. To initialise the pheromone matrix (ξ), we consider equal amount of pheromone on all edges in $\mathcal{E}$: $\xi_{i,j,d} = 1/2$. We restrict each $\xi_{i,j,d}$ in the interval $\left[\frac{1}{|\mathcal{C}|}, 1 - \frac{1}{|\mathcal{C}|}\right]$ [NSW09]. MMAS generates a colony contains artificial ants that each ant performs a random walk to construct a solution step by step. Similar to RGA, ants choose components of their solution at each step from a candidate of available cuts in the neighbourhood of the reclaimer's position. The probability distribution that each ant uses to choose a successor (i.e. the next cut) while constructing a solution is

$$p(c_j|\mathcal{N}_t) = \frac{[\xi_{i,j,d}]^\alpha [\eta(c_j)]^\beta}{\sum_{c_l \in \mathcal{N}_t} [\xi_{i,l,d}]^\alpha [\eta(c_l)]^\beta}$$

where parameters α and β are MMAS control parameters in selection to balance using pheromone or heuristic information. After all ants finish their independent random walk, it is optional for MMAS to employ a local search on the found solutions.

After all ants construct their solution (at end of a generation), the best ant at the iteration x' (with the best obtained solution at iteration) deposits pheromone on the its solution edges as follows,

$$\xi'_{i,j,d} = \begin{cases} \min\{(1-\rho).\xi_{i,j,d} + \rho, 1 - \frac{1}{|\mathcal{C}|}\} & \text{if } (i,j,d) \in x' \\ \max\{(1-\rho).\xi_{i,j,d}, \frac{1}{|\mathcal{C}|}\} & \text{otherwise.} \end{cases}$$

where $0 < \rho < 1$ denotes the evaporation factor. This procedure repeats until MMAS's termination criterion is met.

6.2.3 MMAS With Local Search

Recall from Section 3.2.2 that local search can help random algorithms to avoid local optima. By combining the ability of random search to explore a wide range of solutions with the ability of local search to make locally-improvement moves, the algorithm can effectively find the global optimum. It can also be used to improve the performance of the random algorithm and make it converge faster.

Local search can be complimentary to MMAS, because MMAS explores the search space coarsely. However local search can help to explore in the neighbourhood of a constructed solution more finely. Therefore, after ants construct a solution, their

Algorithm 6.3: Max-Min Ant System (MMAS)

```
initialise ξ                                   ▷ pheromone values initialisation
generate ants for initial colony π_i ⊂ Π
repeat
    for each ant π_i do
        repeat
            construct a solution x step by step, probabilistically
        until solution is complete
    for each ant ∈ Π* do
        perform local search                                   ▷ optional
    Update best found solution x*
    Update ξ
until MMAS termination criterion met
return x*
```

obtained solution can be a good starting point for local search. We refer interested readers to [LMS03] and Section 3.2.2 for more information.

We employ three well-known operators for permutation search problems namely *swap*, *insert* and *inverse* operators. Considering one solution x, these operators get two components in x and perform a local search on the components' position. The swap operator exchanges two components of the solution; the insert operator shifts the second component ahead of the first component. The inverse operator arranges all components between and including the two components in the opposite order. We investigate the iterative local search in the neighbourhood of all components. We call variants of MMAS integrated with local search operators as MMAS-swp, MMAS-ins, MMAS-inv for swap, insert and inverse operators, respectively.

6.2.4 Pilgrim Step Reclaiming Heuristic (PSRH)

Currently, in practice, human experts determine the reclamation sequence planning often using rules of thumb. For example, they know that high-quality material should be mixed with low-quality material to meet the target quality grade and confine the contaminants as close as possible to the limit range. However, human planning is subject to error due to the complexities in the stockyard and subject to multiple operational restrictions. Moreover, human planning is a limited decision-making procedure where it is hard to foresee upcoming stockpile recovery scheduling requests. As a result, human planning can lead to poor reclaiming sequences where it incurs penalty fees, unexpected losses in practice, and perturbations in stockyard management.

Pilgrim step reclaiming is typical reclaiming method in stockpile management. It refers to a method that involves reclaiming material by cutting a series of terraces into it. Each terrace, or bench, is partially reclaimed before the next one is started. The subsequent benches must end before the one above it to prevent the material from collapsing. This method allows for better management of the space where the material is stored and can help avoid some issues with the material being separated, but it may result in a slight decrease in the efficiency of the reclaiming process.

To evaluate Randomised greedy algorithms, MMAS and iterative local search in stockpile recovery, we employ a heuristic representing a rule of thumb that is used for manual planning of the reclaiming sequence in practice. Pilgrim Step Reclaiming Heuristic (PSRH) reclaims one stockpile completely from one end to another. First,

it reclaims the top bench partially, then the lower bench for the reclaimed cut and reclamation repeats till one end is completely reclaimed. Next, the reclaimer machine proceeds and reclaims the rest of the stockpile alike to previous steps from the top bench to the bottom. For example, the following sequence follows PSRH for the first stockpile: We show a solution x obtained by PSRH by the position of cuts as follows.

$$\{(1\text{-}1\text{-}1), (1\text{-}2\text{-}1), (1\text{-}3\text{-}1), (1\text{-}4\text{-}1), (1\text{-}1\text{-}2), \ldots, (1\text{-}4\text{-}2), \ldots, (1\text{-}4\text{-}10)\}$$

6.3 Experimental Setup

In this section, we describe our experimental setup and the assumptions. We consider a real-world stockyard model provided by our industrial partner. This stockpile (as shown in Figure 6.1 has been created using the GPS information of trucks while they stacked the stockpile. There is one machine that can reclaim cuts in one direction, which is South to North. This means that a machine can reclaim only in one direction but can move backward to perform a reclamation. The machine can only process one reclamation task at a time, and the task should be completed before performing the next reclaiming task. For scenarios 2 and 3, we assume that the required tonnage for each delivery is 100,000 tonnes.

DGA is deterministic, and there is no parameter to configure. However, RGA has the parameter λ to control the amount of greediness of the algorithm. We set $\lambda = \{1, 3, 5, 7, 10, 15, 20\}$ and we name the RGA variants as RGA-λ. For MMAS, we set $\alpha = 1$, $\beta = 2$, $\rho = 0.5$ and Π^* only contains best ant in the iteration. We also consider ten ants, and the termination criterion for ACO is when 1000 generations elapsed. With respect to the scenario, a solution construction continues until the solution is complete for all algorithms.

For the randomised algorithms, we run each for 50 times to evaluate them fairly. We also carry out statistical comparisons for randomised algorithms by the Kruskal-Wallis test with a 95% confidence interval integrated with the posteriori Bonferroni test for pair-wise comparisons [CF14]. We rank the obtained solutions by the objective function's lexicographic order to perform the statistical test, and we use their ranks for the statistical test.

For a closer look, we report the median, best and worst solutions obtained by the algorithms in corresponding tables. Note that for PSRH and DGA as they are deterministic algorithms, we report the same value. We also evaluate algorithms by success rate. This measure is the percentage of success for algorithms in obtaining a zero-violation solution. For deterministic algorithms, it could be 0 or 1, but we calculate the fraction out of 50 runs for randomised algorithms.

The implemented framework is available on `https://bit.ly/3AIJVHJ`. The framework includes DGA, RGA, MMAS with and without local search operators. The framework also includes instructions on how to run the code.

6.4 Results And Discussion

Figure 6.2 provides a summary of the results by showing the significance for all scenarios and experiments. Among other, we observe that RGA and MMAS behave differently, while within each group the results are not always different.

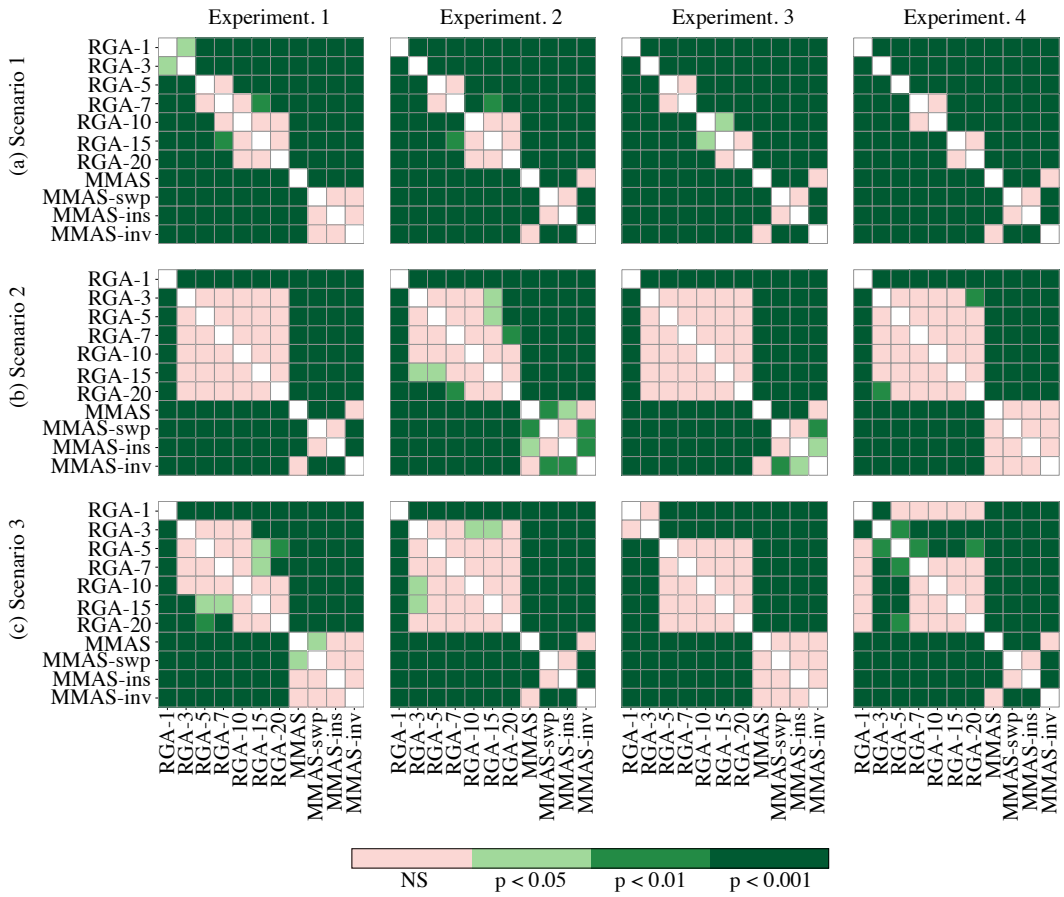

FIGURE 6.2. Significance plot of statistical test for randomised algorithms. p denotes the p-value and NS refers to no significant difference.

TABLE 6.1. Fitness values obtained for the optimised solutions in Scenarios 1 and 2

Instance		PSRH	DGA	best RGA	MMAS	MMAS-swp	MMAS-ins	MMAS-inv
SC1-1	Median	(0.0, 24.6291)	(0.0, 23.685)	(0.0, 24.0001)	(0.0, 23.1543)	**(0.0, 23.0988)**	(0.0, 23.1042)	**(0.0, 23.0988)**
	Best	(0.0, 24.6291)	(0.0, 23.685)	(0.0, 23.3787)	(0.0, 23.1502)	**(0.0, 23.0472)**	(0.0, 23.0504)	**(0.0, 23.0472)**
	Worst	(0.0, 24.6291)	(0.0, 23.685)	(0.0, 24.8425)	(0.0, 23.1661)	**(0.0, 23.1021)**	(0.0, 23.1535)	(0.0, 23.1229)
	Success rate	1	1	1	1	1	1	1
SC1-2	Median	(0.0, 49.6518)	(0.0, 48.1309)	(0.0, 48.1309)	(0.0, 46.6986)	(0.0, 46.4339)	(0.0, 46.6783)	**(0.0, 46.4335)**
	Best	(0.0, 49.6518)	(0.0, 48.1309)	(0.0, 47.7301)	(0.0, 46.598)	(0.0, 46.4005)	(0.0, 46.5274)	**(0.0, 46.3633)**
	Worst	(0.0, 49.6518)	(0.0, 48.1309)	(0.0, 48.3392)	(0.0, 46.8059)	(0.0, 46.4774)	(0.0, 46.7635)	**(0.0, 46.4756)**
	Success rate	1	1	1	1	1	1	1
SC1-3	Median	(0.0, 74.6085)	(0.0, 72.1901)	(0.0, 72.4698)	(0.0, 70.0757)	**(0.0, 69.7819)**	(0.0, 70.0914)	(0.0, 69.7826)
	Best	(0.0, 74.6085)	(0.0, 72.1901)	(0.0, 71.8263)	(0.0, 69.8758)	**(0.0, 69.501)**	(0.0, 69.921)	(0.0, 69.54)
	Worst	(0.0, 74.6085)	(0.0, 72.1901)	(0.0, 76.5797)	(0.0, 70.289)	**(0.0, 69.9158)**	(0.0, 70.2281)	(0.0, 69.9478)
	Success rate	1	1	1	1	1	1	1
SC1-4	Median	(0.0, 99.5444)	(0.0, 96.2366)	(0.0, 96.6943)	(0.0, 93.7738)	**(0.0, 93.1774)**	(0.0, 93.8173)	(0.0, 93.2369)
	Best	(0.0, 99.5444)	(0.0, 96.2366)	(0.0, 95.8881)	(0.0, 93.4327)	(0.0, 92.9711)	(0.0, 93.4237)	**(0.0, 92.9587)**
	Worst	(0.0, 99.5444)	(0.0, 96.2366)	(0.0, 106.4864)	(0.0, 94.3409)	(0.0, 93.497)	(0.0, 94.1342)	**(0.0, 93.4529)**
	Success rate	1	1	1	1	1	1	1
SC2-1	Median	(0.0, 15.9225)	(0.0, 17.4546)	(0.0, 19.4966)	**(0.0, 15.2188)**	(0.0, 17.2239)	(0.0, 15.2251)	(0.0, 17.1897)
	Best	(0.0, 15.9225)	(0.0, 17.4546)	(0.0, 15.4309)	(0.0, 15.1747)	(0.0, 15.1392)	**(0.0, 15.0877)**	**(0.0, 15.0877)**
	Worst	(0.0, 15.9225)	(0.0, 17.4546)	(0.0, 29.3699)	**(0.0, 15.2474)**	(0.0, 19.6106)	(0.0, 15.2504)	(0.0, 19.1453)
	Success rate	1	1	1	1	1	1	1
SC2-2	Median	(0.0, 34.7849)	(0.0, 41.1948)	(0.0, 37.5494)	(0.0, 31.8562)	(0.0, 34.427)	**(0.0, 31.8151)**	(0.0, 34.4242)
	Best	(0.0, 34.7849)	(0.0, 41.1948)	(0.0, 31.2005)	(0.0, 31.218)	(0.0, 33.5467)	**(0.0, 31.0527)**	(0.0, 33.4021)
	Worst	(0.0, 34.7849)	(0.0, 41.1948)	(0.0, 48.4085)	(0.0, 36.0702)	(0.0, 35.6118)	(0.0, 35.8257)	**(0.0, 35.3716)**
	Success rate	1	1	1	1	1	1	1
SC2-3	Median	(0.0304, 49.6518)	(0.1094, 65.5448)	(0.0743, 61.5351)	(0.0, 51.3597)	(0.0, 51.2627)	(0.0, 51.3262)	**(0.0, 51.1625)**
	Best	(0.0304, 49.6518)	(0.1094, 65.5448)	(0.0, 49.9985)	(0.0, 48.9855)	(0.0, 48.906)	(0.0, 49.4875)	**(0.0, 48.3031)**
	Worst	(0.0304, 49.6518)	(0.1094, 65.5448)	(0.1953, 64.7137)	(0.0, 52.7509)	**(0.0, 52.2798)**	(0.0, 52.6403)	(0.0, 52.403)
	Success rate	0	0	0.36	1	1	1	1
SC2-4	Median	(0.1297, 65.5358)	(0.2492, 85.2887)	(0.135, 93.8736)	(0.0, 67.107)	**(0.0, 66.7176)**	(0.0, 67.1275)	(0.0, 67.0235)
	Best	(0.1297, 65.5358)	(0.2492, 85.2887)	(0.0, 72.2375)	(0.0, 65.3268)	**(0.0, 63.1248)**	(0.0, 64.164)	(0.0, 64.8839)
	Worst	(0.1297, 65.5358)	(0.2492, 85.2887)	(0.276, 93.0093)	(0.0, 68.4933)	(0.0, 68.0874)	**(0.0, 68.0108)**	(0.0, 68.0841)
	Success rate	0	0	0.1	1	1	1	1

TABLE 6.2. Fitness values obtained for the optimised solutions by RGA variants in Scenarios 1 and 2

Instance		RGA-1	RGA-3	RGA-5	RGA-7	RGA-10	RGA-15	RGA-20
SC1-1	Median	(0.0, 24.5438)	(0.0, 24.3839)	(0.0, 24.0001)	(0.0, 23.9599)	(0.0, 23.9059)	(0.0, 23.7538)	(0.0, 23.685)
	Best	(0.0, 23.8067)	(0.0, 23.6249)	(0.0, 23.3787)	(0.0, 23.555)	(0.0, 23.6736)	(0.0, 23.685)	(0.0, 23.685)
	Worst	(0.0, 25.5281)	(0.0, 25.1717)	(0.0, 24.8425)	(0.0, 24.9514)	(0.0, 24.184)	(0.0, 24.0823)	(0.0, 23.8764)
	Success rate	1	1	1	1	1	1	1
SC1-2	Median	(0.0, 53.0194)	(0.0, 51.4209)	(0.0, 49.7899)	(0.0, 48.3997)	(0.0, 48.3136)	(0.0, 48.225)	(0.0, 48.1309)
	Best	(0.0, 50.1211)	(0.0, 48.2598)	(0.0, 47.9263)	(0.0, 47.857)	(0.0, 47.8382)	(0.0, 47.7301)	(0.0, 47.7301)
	Worst	(0.0, 55.9081)	(0.0, 55.2465)	(0.0, 53.7038)	(0.0, 51.2356)	(0.0, 51.3081)	(0.0, 48.5025)	(0.0, 48.3392)
	Success rate	1	1	1	1	1	1	1
SC1-3	Median	(0.0, 91.2078)	(0.0, 85.4423)	(0.0, 80.245)	(0.0, 76.2518)	(0.0, 73.5349)	(0.0, 72.7143)	(0.0, 72.4698)
	Best	(0.0, 82.3909)	(0.0, 77.8463)	(0.0, 73.6328)	(0.0, 72.8549)	(0.0, 72.3544)	(0.0, 71.9452)	(0.0, 71.8263)
	Worst	(0.0, 97.2994)	(0.0, 92.4979)	(0.0, 87.7998)	(0.0, 83.3502)	(0.0, 81.6265)	(0.0, 78.0395)	(0.0, 76.5797)
	Success rate	1	1	1	1	1	1	1
SC1-4	Median	(0.0, 132.2599)	(0.0, 121.5268)	(0.0, 114.4084)	(0.0, 108.017)	(0.0, 103.2114)	(0.0, 97.8469)	(0.0, 96.6943)
	Best	(0.0, 121.8832)	(0.0, 111.9097)	(0.0, 98.3846)	(0.0, 97.5143)	(0.0, 97.1517)	(0.0, 95.9042)	(0.0, 95.8881)
	Worst	(0.0, 141.1381)	(0.0, 132.5836)	(0.0, 130.0943)	(0.0, 119.5976)	(0.0, 112.6485)	(0.0, 109.542)	(0.0, 106.4864)
	Success rate	1	1	1	1	1	1	1
SC2-1	Median	(0.0, 40.5356)	(0.0, 30.954)	(0.0, 25.0367)	(0.0, 22.8501)	(0.0, 19.4966)	(0.0, 18.1133)	(0.0, 17.4546)
	Best	(0.0, 28.6118)	(0.0, 21.4039)	(0.0, 15.6873)	(0.0, 15.6881)	(0.0, 15.4309)	(0.0, 15.8339)	(0.0, 15.6627)
	Worst	(0.0338, 56.0964)	(0.0, 42.0807)	(0.0, 35.6851)	(0.0, 30.6913)	(0.0, 29.3699)	(0.0, 25.5533)	(0.0, 23.3973)
	Success rate	0.94	1	1	1	1	1	1
SC2-2	Median	(0.0, 59.487)	(0.0, 51.2316)	(0.0, 44.3295)	(0.0, 38.9105)	(0.0, 37.5494)	(0.0, 39.1054)	(0.0, 40.9076)
	Best	(0.0, 42.7386)	(0.0, 39.6749)	(0.0, 34.8443)	(0.0, 31.7795)	(0.0, 31.2005)	(0.0, 34.1988)	(0.0, 31.5022)
	Worst	(0.0299, 61.5492)	(0.0468, 58.626)	(0.0, 57.745)	(0.0968, 53.9373)	(0.0, 48.4085)	(0.0958, 57.4942)	(0.1966, 49.9642)
	Success rate	0.96	0.98	1	0.96	1	0.94	0.9
SC2-3	Median	(0.0, 81.813)	(0.0, 73.3463)	(0.0, 70.4631)	(0.0672, 71.3798)	(0.0743, 61.5351)	(0.077, 66.7136)	(0.0881, 74.3815)
	Best	(0.0, 69.0095)	(0.0, 57.2619)	(0.0, 52.3124)	(0.0, 50.814)	(0.0, 49.9985)	(0.0, 50.523)	(0.0, 50.1781)
	Worst	(0.0875, 85.7634)	(0.1427, 78.1663)	(0.1684, 83.8003)	(0.1608, 77.6216)	(0.1953, 64.7137)	(0.1785, 75.263)	(0.1838, 67.9059)
	Success rate	0.8	0.76	0.54	0.36	0.36	0.22	0.08
SC2-4	Median	(0.0488, 96.8375)	(0.09, 86.8199)	(0.135, 93.8736)	(0.1659, 78.5034)	(0.1894, 83.6359)	(0.1969, 92.1301)	(0.1775, 85.96)
	Best	(0.0, 82.3783)	(0.0, 81.812)	(0.0, 72.2375)	(0.0, 72.8784)	(0.0, 77.8025)	(0.0549, 80.3513)	(0.0, 75.7459)
	Worst	(0.1637, 95.7115)	(0.2405, 101.1136)	(0.276, 93.0093)	(0.2612, 89.2846)	(0.2795, 84.3993)	(0.2622, 98.1771)	(0.2765, 85.348)
	Success rate	0.34	0.14	0.1	0.04	0.02	0	0.02

6.4.1 Scenario 1

Here, we assume that the problem aim is reclaiming the whole stockyard. Table 6.1 lists the obtained objectives for Scenarios 1 and 2. Note that we only report the best RGA variant for each experiment for brevity. The complete information regarding the outcomes from the RGA variants can be found in Table 6.2.

For all experiments, we observe that all algorithms can obtain a success rate of 100%. Therefore, for the evaluation, we only look at the utility cost. We can also see that DGA outperforms PSRH, meaning that acting deterministic greedy is better than using PSRH for this scenario. We observe the same pattern for RGA, but with there is a statistical difference for various experiments. We can see for all experiments that RGA-15 and RGA-20 are not significantly different.

RGA-1, RGA-3, RGA-5 can find solutions with large utility, as the worst obtained utility among these variants are 25.5281, 25.1717, 24.8425, respectively. We see that as λ increases, the median value of utility decreases, where it leads to better solutions. We see for experiments 2-4, RGA-20 can obtain the best solution with the utility of 47.7301, 71.8263, and 95.8881, respectively. However, for experiment 1, which is the most straightforward instance in our study, RGA-5 obtains the best solution with a utility of 23.3787, but with a higher median than the RGA variant where $\lambda \geq 7$.

We also observe that the best solution obtained by RGA is the same as DGA in experiment 1. However, as the stockyard's size becomes larger, adding randomness results in RGA outperforming DGA. To determine the best RGA variant of experiments 2-4, we report RGA-20, and for experiment 1. and we report RGA-5 for experiment 1.

We see that all MMAS variants with or without local search outperform PSRH, DGA and RGA variants. For experiment 1, the local search variants outperform those without. Moreover, there is no significant difference among the local search variants. However, for experiments 2-4, we see MMAS-swp and MMAS-ins are significantly different and better than other variants. Nonetheless, there exists no significant difference between MMAS and MMAS-inv.

6.4.2 Scenario 2

In this scenario, we aim to plan the reclamation to fulfill deliveries. We observe that PRSH and DGA can obtain a zero violation solution for the first two experiments. However, for the last two experiments, their obtained solutions are a non-zero violation solution. Moreover, DGA obtains worse solutions than PSRH. For this scenario, we can see that acting greedy by DGA leads to weaker solutions, unlike scenario 1 when the problem is more similar to real-world conditions.

For all experiments, among RGA variants, we see no significant difference where $3 \leq \lambda \leq 10$. Moreover, in experiment 1, we observe that the RGA-1 success rate is 94%; however, other RGA variants are 100% successful. For experiment 1, we report RGA-10, where it can obtain the best solution among RGA variants. For experiment 2, we see that RGA-1, RGA-3, RGA-7, RGA-15, and RGA-20 success rates are at least 90%. Among RGA-10, RGA-15, and RGA-20 where all are non-significant different from each other, we report RGA-10 because its success rate is 100% and it can obtain reasonably good solutions.

Experiments 3 and 4 are more challenging instances because we can see that as the number of deliveries increases, the resources in the stockyard are close to being exhausted. It makes it harder for the algorithm to provide non-zero violations for all the deliveries. For experiment 3, we observe that among RGA variants, as λ increases, the success rate decreases from 80% to 8%. We report RGA-10 as the best

RGA variant for experiment 3. For experiment 4, the success rate for RGA variants decreases from 34% to 0%, where RGA-15 success rate is zero. We report RGA-5 as the best RGA variant where it can find the best solution (0.0, 72.2375).

Similar to experiments 2-4 in scenario 1, we see no significant difference between MMAS and MMAS-inv. We also observe that MMAS with or without local search outperforms PSRH, DGA and RGA variants. The success rate for all MMAS variants is 100%, where it enables us to only look at their utility function to compare them more easily. We can see that in experiments 1 and 2, MMAS-ins outperforms MMAS-swp. However, in experiments 3 and 4, with more complicated instances, MMAS-swp is the best among the two. However, in experiment 4, as the most challenging experiment in this scenario, all MMAS variants are not significantly different.

6.4.3 Scenario 3

This scenario is the most challenging one and the most similar to the real-world application which considers window target quality. Table 6.3 lists the results. Note that The complete information regarding the outcomes from the RGA variants can be found in Table 6.4. In all experiments, we observe that PSRH and DGA obtain non-zero violation solutions where DGA can obtain a better solution. Among RGA variants for experiments 1-3, we see that variants with $3 \leq \lambda \leq 7$ are non-significant different from each other, and the same behavior applies to the variants with $10 \leq \lambda \leq 20$.

RGA-3 and RGA-5 obtain solutions with a higher partial success rate. Similar to other scenarios, MMAS and its variants outperform other algorithms, and for experiments 1-3, all show a 100% success rate. However, for experiment 4, we see an occasional success for only the MMAS variants. This could be because of the way the experiments were defined: almost all resources are exhausted, and it is hard to find a solution without violating the quality objectives.

For experiment 1, RGA-10, RGA-15 and RGA-20 obtain their best solutions with zero violation with utility of 17.0583, 17.1489 and 16.714 with algorithm success rates of 68%, 44% and 42% respectively. Similarly, for experiment 2, RGA-5, RGA-7, RGA-20 obtain their best solutions as zero-violation with utility of 34.3946, 34.6121 and 34.332 with their success rates as 42%, 20% and 2% respectively. It shows that for these complex scenarios in both experiments, some RGA variants are able to find better solutions than MMAS but with a lower success rate. It shows that there exists room for improvement of MMAS variants with a proper tuning of MMAS and improve MMAS ability to explore the search space better. For experiment 1-3, we report RGA-20 as the best RGA variant.

Experiment 4 of this scenario is the most difficult instance for our study. We can see that all RGA variants are unsuccessful. However, all MMAS variants can obtain a zero-violation solution but note that the success rate of MMAS is low. It demonstrates that MMAS and its variants can find a good solution. However, the algorithm's quality for robustness in finding zero-violation solutions in multiple runs could be improved. For experiment 4, we report RGA-3 as the best RGA variant. It can obtain a solution where $v_1(x) = 0$ as (0.0, 0.3736, 84.114) and it is the best solution among RGA variants.

6.5 Conclusions

We stated the stockpile recovery problem as a combinatorial optimisation problem with a lexicographic objective function considering technical restrictions in practice.

TABLE 6.3. Fitness values obtained for the optimised solutions in Scenario 3

Instance		PSRH	DGA	best RGA	MMAS	MMAS-swp	MMAS-ins	MMAS-inv
SC3-1	Median	(0.0, 0.0801, 15.9225)	(0.0, 0.0033, 31.9449)	(0.0, 0.0, 26.1112)	**(0.0, 0.0, 21.6272)**	(0.0, 0.0, 22.5023)	(0.0, 0.0, 22.3799)	(0.0, 0.0, 22.4573)
	Best	(0.0, 0.0801, 15.9225)	(0.0, 0.0033, 31.9449)	**(0.0, 0.0, 16.714)**	(0.0, 0.0, 17.5283)	(0.0, 0.0, 20.5082)	(0.0, 0.0, 20.7674)	(0.0, 0.0, 19.6313)
	Worst	(0.0, 0.0801, 15.9225)	(0.0, 0.0033, 31.9449)	(0.0, 0.0, 27.6398)	**(0.0, 0.0, 22.8708)**	(0.0, 0.0, 23.553)	(0.0, 0.0, 23.0829)	(0.0, 0.0, 23.393)
	Success rate	0	0	0.42	1	1	1	1
SC3-2	Median	(0.0, 0.2286, 34.7849)	(0.0, 0.0668, 54.3735)	(0.0, 0.0, 43.803)	(0.0, 0.0, 38.6463)	**(0.0, 0.0, 37.7133)**	(0.0, 0.0, 38.5856)	(0.0, 0.0, 37.966)
	Best	(0.0, 0.2286, 34.7849)	(0.0, 0.0668, 54.3735)	**(0.0, 0.0, 34.332)**	(0.0, 0.0, 35.2312)	(0.0, 0.0, 36.1584)	(0.0, 0.0, 36.916)	(0.0, 0.0, 36.012)
	Worst	(0.0, 0.2286, 34.7849)	(0.0, 0.0668, 54.3735)	(0.062, 0.0, 47.5242)	(0.0, 0.0, 41.0108)	(0.0, 0.0, 40.1809)	(0.0, 0.0, 39.9312)	**(0.0, 0.0, 39.6312)**
	Success rate	0	0	0.02	1	1	1	1
SC3-3	Median	(0.0304, 0.4242, 49.6518)	(0.0, 0.1015, 70.2569)	(0.0496, 0.0489, 63.8654)	(0.0, 0.0, 59.6043)	**(0.0, 0.0, 55.9089)**	(0.0, 0.0, 58.6546)	(0.0, 0.0, 56.0387)
	Best	(0.0304, 0.4242, 49.6518)	(0.0, 0.1015, 70.2569)	(0.0, 0.0489, 57.0279)	(0.0, 0.0, 56.3588)	**(0.0, 0.0, 53.7049)**	(0.0, 0.0, 56.568)	(0.0, 0.0, 54.2217)
	Worst	(0.0304, 0.4242, 49.6518)	(0.0, 0.1015, 70.2569)	(0.1295, 0.0489, 79.1781)	(0.0, 0.0, 62.2242)	**(0.0, 0.0, 57.2936)**	(0.0, 0.0, 59.7979)	(0.0, 0.0, 57.5471)
	Success rate	0	0	0	1	1	1	1
SC3-4	Median	(0.1297, 2.0262, 65.5358)	(0.0627, 0.7477, 84.094)	(0.126, 0.3736, 82.627)	(0.0, 0.0064, 81.1742)	(0.0, 0.0, 80.9867)	(0.0, 0.0281, 79.3744)	**(0.0, 0.0, 76.3152)**
	Best	(0.1297, 2.0262, 65.5358)	(0.0627, 0.7477, 84.094)	(0.0, 0.3736, 84.114)	(0.0, 0.0064, 88.1848)	(0.0, 0.0, 81.6204)	(0.0, 0.0281, 82.8373)	**(0.0, 0.0, 72.6274)**
	Worst	(0.1297, 2.0262, 65.5358)	(0.0627, 0.7477, 84.094)	(0.2251, 0.3736, 97.6799)	(0.0, 0.0064, 80.9308)	**(0.0, 0.0, 77.0153)**	(0.0, 0.0281, 82.1064)	(0.0, 0.0, 83.4832)
	Success rate	0	0	0	0	0.02	0	0.1

TABLE 6.4. Fitness values obtained for the optimised solutions by RGA variants in Scenario 3

Instance		RGA-1	RGA-3	RGA-5	RGA-7	RGA-10	RGA-15	RGA-20
SC3-1	Median	(0.0, 0.0, 32.241)	(0.0, 0.0, 30.1275)	(0.0, 0.0, 26.5095)	(0.0, 0.0, 23.5585)	(0.0, 0.0, 22.4319)	(0.0, 0.0, 23.4121)	(0.0, 0.0, 26.1112)
	Best	(0.0, 0.0, 29.8447)	(0.0, 0.0, 21.8285)	(0.0, 0.0, 19.5892)	(0.0, 0.0, 17.6229)	(0.0, 0.0, 17.0583)	(0.0, 0.0, 17.1489)	(0.0, 0.0, 16.714)
	Worst	(0.0074, 0.0, 39.809)	(0.0, 0.0, 39.6559)	(0.0, 0.0, 37.8551)	(0.0, 0.0, 25.1639)	(0.0, 0.0, 33.0818)	(0.0, 0.0, 29.8802)	(0.0, 0.0, 27.6398)
	Success rate	0.2	0.9	0.92	0.88	0.68	0.44	0.42
SC3-2	Median	(0.0, 0.0, 54.8851)	(0.0, 0.0, 52.2506)	(0.0, 0.0, 44.3513)	(0.0, 0.0, 38.2791)	(0.0, 0.0, 49.2167)	(0.0, 0.0, 48.9637)	(0.0, 0.0, 43.803)
	Best	(0.0, 0.0, 59.9353)	(0.0, 0.0, 37.0024)	(0.0, 0.0, 34.3946)	(0.0, 0.0, 34.6121)	(0.0, 0.0, 41.6276)	(0.0, 0.0, 40.6691)	(0.0, 0.0, 34.332)
	Worst	(0.0, 0.0, 57.5176)	(0.0663, 0.0, 56.4758)	(0.1536, 0.0, 62.8787)	(0.0798, 0.0, 57.2654)	(0.1113, 0.0, 61.5711)	(0.096, 0.0, 56.4824)	(0.062, 0.0, 47.5242)
	Success rate	0.02	0.62	0.42	0.2	0.02	0.1	0.02
SC3-3	Median	(0.0, 0.0787, 85.3033)	(0.0195, 0.0, 65.288)	(0.0731, 0.0, 79.3199)	(0.0747, 0.1, 72.9863)	(0.0767, 0.1406, 67.0705)	(0.0724, 0.1018, 67.7157)	(0.0496, 0.0489, 63.8654)
	Best	(0.0, 0.0787, 86.8806)	(0.0, 0.0, 62.6515)	(0.0, 0.0, 55.8103)	(0.0, 0.1, 56.7536)	(0.0, 0.1406, 59.9472)	(0.0, 0.1018, 58.5953)	(0.0, 0.0489, 57.0279)
	Worst	(0.0952, 0.0787, 72.9958)	(0.1319, 0.0, 68.798)	(0.1804, 0.0, 82.5686)	(0.1529, 0.1, 81.8514)	(0.1544, 0.1406, 76.734)	(0.1673, 0.1018, 80.8484)	(0.1295, 0.0489, 79.1781)
	Success rate	0	0.1	0.02	0	0	0	0
SC3-4	Median	(0.0798, 0.6426, 93.5596)	(0.126, 0.3736, 82.627)	(0.1836, 1.9421, 88.1446)	(0.1773, 1.4155, 90.8624)	(0.1738, 2.2083, 91.1521)	(0.1641, 1.3447, 88.3056)	(0.1355, 1.1435, 78.1302)
	Best	(0.0, 0.6426, 102.6171)	(0.0, 0.3736, 84.114)	(0.0905, 1.9421, 87.8819)	(0.0752, 1.4155, 80.9774)	(0.0768, 2.2083, 76.0499)	(0.0541, 1.3447, 80.1416)	(0.0612, 1.1435, 81.8911)
	Worst	(0.2223, 0.6426, 86.0725)	(0.2251, 0.3736, 97.6799)	(0.2507, 1.9421, 102.5166)	(0.2475, 1.4155, 87.7491)	(0.229, 2.2083, 94.0736)	(0.2533, 1.3447, 99.2349)	(0.2233, 1.1435, 81.3767)
	Success rate	0	0	0	0	0	0	0

We have used optimisation methods to construct a reclaiming schedule step by step to meet the precedence constraints in solving the problem. For this purpose, we have explored the use of deterministic and randomised greedy algorithms. We have also employed MMAS as a variant of the ACO algorithm, and we considered three local search operators for it, namely swap, insert, and inversion. We investigated multiple scenarios and experiments. In our experiments, which have covered various real-world aspects, We have developed multiple scenarios and experiments to investigate our approach and study the problem model considering various complexities and similarities to real-world applications. We have observed that adding randomness to the deterministic greedy algorithm has helped it find better solutions when the scenario is more similar to the real-world problem. We have also seen that MMAS, especially with a local search, outperforms other algorithms with a high success rate. Among the local search operators, the insert operator is not significantly different from MMAS without the local search. However, swap and inverse operators can help to find better solutions. We have used several default parameters for the algorithms. For future studies, it could be interesting (1) to investigate automated online/offline tuning of the parameters of algorithms, (2) to study other ACO variants, and (3) to develop local search operators that are specific to this reclamation problem.

Chapter 7

Run-of-Mine Stockyard Recovery Scheduling and Optimisation for Multiple Reclaimers

We discussed in the previous chapter the stockpile recovery problem considering a single reclaimer. This chapter extends the study for a stockyard recovery problem scheduling using multiple reclaimers. We aim to develop an effecting reclaiming schedule considering the quality objectives as discussed in the previous chapter and enable long-term delivery planning using multiple reclaimers.

In Chapter 6, we considered the stockpile recovery problem with separate cuts instead of a whole bulk item. We modelled the stockpile recovery scheduling problem by considering realistic settings. We defined the problem as a combinatorial optimisation problem with precedence constraints to address the inaccessibility issue of the cuts in the stockyard. We introduced a lexicographic objective function to prioritise penalty fees over operation costs with respect to the end-user preferences.

The main disadvantage of this initial study was that it only considers a single reclaimer machine in the stockyard; however, stockyards are reclaimed in parallel using multiple reclaimers in practice to prepare deliveries. Another assumption is that we only considered a single type of reclaimer with a single mechanical direction of reclamation. However, in practice, the types of reclaimers can be different, and the variety can bring up different reclaiming directions using their mechanical arms. In the previous chapter, our goal was to schedule reclaiming for up to four deliveries from four stockpiles containing the same type of material. We used solution construction based methods such as greedy algorithms and ant colony optimisation with local search to build a valid solution step by step. We showed that deterministic greedy algorithms might fail to obtain good solutions when the number of upcoming deliveries grows. However, ant colony optimisation with local search can be a viable solver to these complex problems.

In this chapter, we investigate the stockpile recovery scheduling problem using multiple reclaimers with more realistic settings than before. We define the stockyard as a directed graph with cuts as vertices and direction as the mechanical *reclaiming direction*. The reclaiming direction is an inherent element of a reclaimer machine; for example, front end loaders can only reclaim in one way, while bucket wheel reclaimers can reclaim in two directions. By combining GPS data from the haul trucks and laser scanning, our industry partner can model the stockpiles divided into smaller cuts with their own quality characteristics instead of considering them as one large pile with average quality. We consider the problem regardless of stockyard economic minerals or reclaimer machines to reach a scalable method. We schedule multiple reclaimers in the stockyard in order to provide deliveries sequentially. To avoid penalty fees and

reduce operational costs, we aspire to achieve a good schedule for the reclaimers while preserving the target quality of deliveries.

With multiple reclaimers in this chapter, we must consider their interaction so that when they work in parallel, they do not come too close to one another for safety constraints. The schedule for a single reclaimer represents the sequence of cuts to be reclaimed regardless of time. In comparison, schedules are obtained while considering multiple reclaimers that they reclaim cuts at a given time step according to the reclaimers' reclaiming directions.

These methods require a solution construction heuristic. In this study, to simulate the interaction among reclaimers and ensure the safety distancing, we develop a custom solution construction heuristic to consider all reclaimers in preparing a delivery simultaneously and sequentially. To select a job at each step, we employ selection strategies from deterministic and randomised greedy algorithms, indicated as DGA and RGA, respectively, and Max-Min Ant System (MMAS) with a customised local search to tackle the problem. Our experiments consider realistic stockpile recovery settings integrating different types of reclaimers, material, and interaction among the reclaimers.

This chapter is based on the work published in 37th Annual ACM Symposium on Applied Computing. ACM, 2022 [Ass+22a]. We added more experiments and the rest of this chapter is organised as follows. We define the stockpile recovery problem using multiple reclaimers before explaining the objective function. Next, we present the optimisation algorithms and solution construction heuristic for simulating the reclaimers interactions. Then, we set up experiments, fine-tune our algorithms and report on the behaviour and quality of obtained solutions for each problem instance. We find that MMAS with local search can outperform other methods in most of the case studies. Finally, we finish with some concluding remarks and some suggestions for future work.

7.1 Problem Statement

We discussed the stockpile recovery problem in Chapter 6 using single reclaimer. In this section we extend the single reclaimer problem to cover a stockyard with multiple reclaimers and we change the problem statement, constraints and objective functions accordingly.

Similar to the previous Chapter, we model a given stockyard in form of a directed graph $\mathcal{G} = (\mathcal{C}, \mathcal{E})$ without cycles where $\mathcal{C} = \{c_1, c_2, \ldots, c_J\}$ is the set of cuts in the stockyard including J cuts with index j. Each cut denotes a slice of a stockpile at a certain location in the stockyard with respect to the positioning of its bench, stockpile and row in the stockyard. Each cut contains information on the mineral compositions of economic and contaminant minerals, denoted by $m_{j,k}$ where $k = \{1, 2, \ldots, \mathcal{K}\}$, which defines the set of chemical elements preferred by the end-user to be evaluated.

Operational constraints include the accessibility of cuts in the stockyard where the lower benches are unreachable before reclaiming the higher bench. Furthermore, if multiple stockpiles are present, the reclaimers should keep a safe distance between themselves to avoid collisions.

Figure 7.1 shows the stockyard configuration in our problem for stockyard recovery using multiple reclaimers. Note that the row 1 shown in this figure is the same row we studied in Chapter 6 and shown in Figure 6.1. Similar to Chapter 6 each stockpile can be split into 4 benches, each of which has 10 cuts. However, we have four rows, each one containing four stockpiles. Some reclaimers, such as bucket wheel reclaimers,

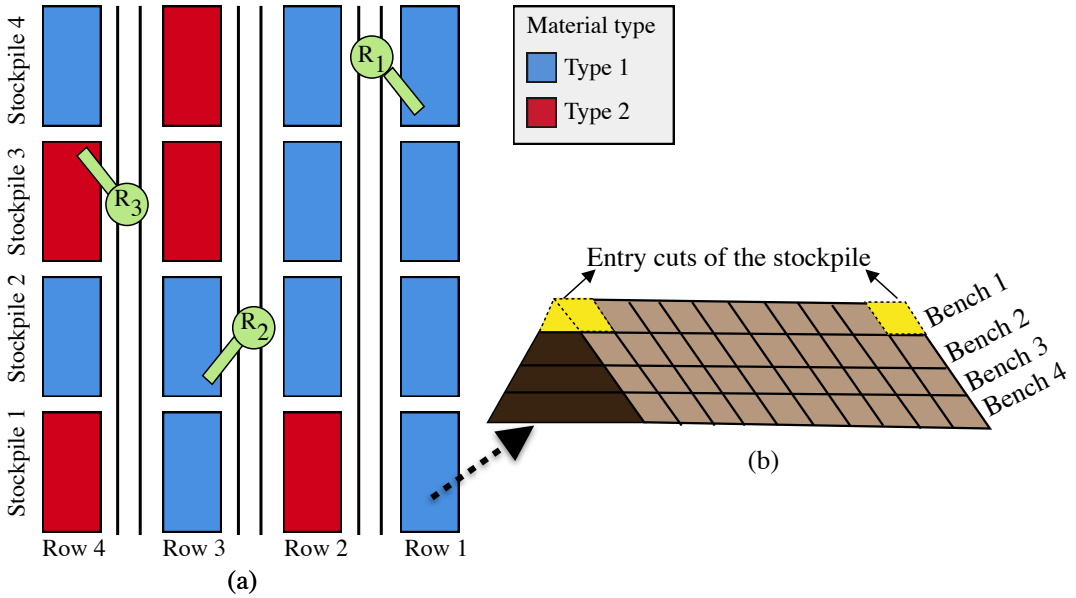

FIGURE 7.1. (a) Top view of the stockyard configuration (b) Layout of a single stockpile with four benches each containing ten cuts

can move their mechanical arm in two different directions during recovery, which we refer to as reclaiming direction (ϕ). We suppose there are two kinds of deliveries in terms of the particle size distribution of stacked material. Each stockpile can only contain one type of material. Entry cuts refer to the first cut of the stockpile that can be accessed with a reclaimer with a specific reclaiming direction. We consider that a reclaimer can operate on each row and they are parallel to each other.

Each cut (c_j) has a specified tonnage given by Γ_j, and reclaiming a cut takes a specific amount of time proportionate to its size denoted by τ_j. $\mathcal{E}$ denotes the set of edges representing the immediate predecessors for each cut with its corresponding reclaiming direction. $e_{j,\phi_j} \in \mathcal{E}$ denotes the edge connecting to the c_j with reclaiming direction of ϕ_j; the edge physically represents the reclaimer *job* where reclaimer relocates from its current position to c_j and reclaims the destination cut with reclaiming direction of ϕ_j. Precedence constraints determine the validity of a schedule. If it fails for a segment of the solution, the schedule becomes unable to be processed.

Stockpile recovery is a time-consuming operation that entails the relocation of reclaimers, such as front-end loaders and bucket wheel reclaimers, in the stockyard and the time it takes to reclaim a cut proportionate to its size. A cut is a section of piled material that can weigh 1000 to 5000 tonnes. Operation time and operation costs are phrases that can be used interchangeably. To calculate a reclaimer's relocation cost shown as $\mathcal{T}_{i,j,\phi_i,\phi_j}$, we need to know its current position at cut i, the destination (j), and their corresponding reclaiming directions, ϕ_i and ϕ_j, respectively.

A candidate solution for a schedule, x represents how jobs are scheduled for reclaimers. Sets R and D denote the reclaimer and delivery sets, respectively. We show a sub-segment of x for reclaimer r in preparation of delivery d as x_d^r.

We aim to maintain the target quality of deliveries in terms of economic and contaminant minerals while lowering operating costs based on end-user preferences. The first objective is to guarantee that the average of chemical minerals in delivery remains within a predefined range depending on the type of material in delivery. We calculate the average target quality for each mineral (k) with respect to the delivery

d as follows. Note that to form $x_{d,k}$ we consider all reclaimers.

$$\hat{x}_{d,k} = \frac{1}{|x_d|} \sum_{e_{j,\phi_j} \in x_d} m_{j,k}$$

where, the predefined range constraint is

$$\underline{m_{d,k}} \leq \hat{x}_{d,k} \leq \overline{m_{d,k}} \quad \forall k \in K$$

Each mineral has a different magnitude, as well as a lower and maximum limit. As shown below, we utilise the bracket-operator penalty approach [Deb01] to evaluate the degree of violation for *average target quality*.

$$\left\langle \hat{x}_{d,k} \right\rangle = \begin{cases} \frac{|\hat{x}_{d,k} - \overline{m_{d,k}}|}{|\overline{m_{d,k}}|} & \text{if } \hat{x}_{d,k} > \overline{m_{d,k}} \\ \frac{|\hat{x}_{d,k} - \underline{m_{d,k}}|}{|\underline{m_{d,k}}|} & \text{if } \hat{x}_{d,k} < \underline{m_{d,k}} \\ 0 & \underline{m_{d,k}} \leq \hat{x}_{d,k} \leq \overline{m_{d,k}} \end{cases}$$

To calculate the total violation for average target quality in x for all reclaimers for the delivery d we have:

$$v_1(x_d) = \sum_{k=1}^{K} \left\langle \hat{x}_{d,k} \right\rangle$$

Another objective is to ensure that a massive delivery in different packages remains consistent. We refer to it as *window target quality*.

$$\widetilde{x}_{d,j,k} = \frac{m_{j,k} + m_{j-1,k} + m_{j-2,k}}{3}$$

We use the bracket operator, as we did for the first objective, to determine the degree of violation for this objective, and we have:

$$v_2(x_d) = \sum_{k=1}^{K} \sum_{j=4}^{|x_{d,k}|} \left\langle \widetilde{x}_{d,j,k} \right\rangle$$

j refers to the position of a cut in the solution segment since we want to calculate the window target quality after three cuts have already been reclaimed for a delivery. The third objective is to lowering the operation costs, where we want to have a schedule for reclaimers to prepare the deliveries faster. We define a utility function as follows.

$$u(x_d) = \sum_{c_j \in x_d} \frac{T_{e_{j,\phi_j}}}{\Gamma_j}$$

where $T_{e_{j,\phi_j}}$ is the cost required to complete job e_{j,ϕ_j} considering the cost of relocation and cut reclamation.

$$T_{e_{j,\phi_j}} = \mathcal{T}_{i,j,\phi_i,\phi_j} + \tau_j$$

7.1.1 Lexicographic Objective Function

We prioritise target quality over operational costs because if the target quality is violated, the stockpile manager must pay financial penalties. As a result, the primary objective is to avoid violating the average target quality; the second priority is to avoid

violating window target quality and, subsequently, to reduce the utility. We employ a lexicographic objective function for minimisation to consider these priorities.

$$f(x) = \left(\sum_{d=1}^{D} v_1(x_d), \sum_{d=1}^{D} v_2(x_d), \sum_{d=1}^{D} u(x_d) \right)$$

Order in the objective function matters. To compare two solutions x and z, we have:

$$\begin{aligned} & f(x) \leq f(z) \\ \text{iff} \quad & v_1(x) \leq v_1(z) \vee \\ & (v_1(x) = v_1(z) \wedge v_2(x) \leq v_2(z)) \vee \\ & (v_1(x) = v_1(z) \wedge v_2(x) = v_2(z) \wedge u(x) \leq u(z)) \end{aligned}$$

For example, among $f(x) = (0.01, 0, 35)$ and $f(z) = (0, 0.01, 15)$ and $f(w) = (0, 0, 45)$, first, we look at the first component, and we see that v_1 is zero for solutions z and w; thus, solution x is the worst of all solutions. To compare solutions z and w, we consider the second component (v_2) where solution w outperforms the solution z. We have:

$$f(w) < f(z) < f(x)$$

Our problem can be viewed as an extended version of the Travelling Salesperson Problem (TSP) with real-world constraints and objectives. In our case, we replace the cities and TSP distances with mineral cuts and the reclamation job costs, respectively. Furthermore, we have several agents (reclaimers) on graphs that must adhere to a no-cross condition. We aim to find a solution to maintain the objectives with respect to the end-user preferences. There are precedence constraints in accessing the cuts and requirements to maintain the target quality of deliveries.

Note that Both knapsack problem (which represented the primary problem in Chapter 4 and TSP problems are classic combinatorial problems, and they can represent a broad class of optimization problems that arise in many practical applications, such as logistics and supply chain management, manufacturing, and logistics.

In logistics and supply chain management, for example, the knapsack problem is used to model the problem of packing items into containers or vehicles in a way that maximises their value while keeping them within a certain weight limit. This is a common problem that arises in shipping, warehouse management, and other areas of logistics. In manufacturing, the knapsack problem can be used to model the problem of selecting a subset of machine tools or other resources to use in production to maximise the output while minimising the cost. In logistics, the travelling salesman problem is used to model the problem of finding the most efficient route for delivery truck, ships or planes to visit a set of customers or delivery points. This problem is of great importance for companies that need to plan their routes in an efficient way to minimise the distance travelled, fuel consumption, and other expenses.

7.2 Optimisation Methods

In this section, we explain our methods. First, we present the solution construction heuristic, which simulates the reclaimers interactions to construct a schedule. Next, we explain the employed algorithms that generate a valid solution step by step while adhering to the precedence constraints. We look at deterministic and randomised

variants of greedy algorithms and ant colony optimisation with and without the local search.

7.2.1 Solution Construction Heuristic

Algorithm 7.1 shows the solution construction heuristic procedure. This solution construction heuristic is more sophisticated compared to the counterpart in the previous chapter because we need to consider the interaction among the reclaimers. We assume that each reclaimer can only do one reclamation job at a time. Reclaimers cannot be interrupted while performing a reclaiming job, and the job must be completed before starting another (non-preemptive constraint).

Each reclaimer can only reclaim from its adjacent stockpiles. For example, in Figure 7.1, R_1 can only reclaim from Row 1 and Row 2, and all stockpiles in Row 2 are shared between R_1 and R_2. However, reclaimers on a shared row can not get too close to one another in terms of a safety distance constraint.

Reclaimers can be *idle* or *busy* at a time. Initially, the solution set x is empty, and reclaimers are idle. Reclaimers have access only to the entry cuts at their adjacent stockpiles. The entry cuts are positioned opposite with respect to the other reclaiming direction.

We assume that the reclaimers begin with reclaiming the fixed entry cut with the fixed initial direction (ϕ_0) from a stockpile with the material type of first delivery ($d = 1$). We denote the initial reclamation job as $e_{c_0^r,\phi_0}$ where c_0^r is the fixed entry cut for reclaimer r. Reclaimers record a completion time of the job as they become *busy* and we push the cut to their queue. When the initial job's completion time has passed, the reclaimer(s) become idle, and we add the reclaimed cut to the reclaimer schedule. Next, we can figure out what cuts are accessible for a reclaimer. $\mathcal{N}_t^r$ shows the available cuts in the neighbourhood of reclaimer r at its position at time step t. If more than one reclaimer becomes idle at the same time, the total neighbourhood is a set containing candidates from each reclaimer paired. Note that each reclaimer's neighbourhood is made up of a variety of reclaiming directions. The following criteria are used to choose eligible cuts for inclusion in $\mathcal{N}_t$.

- Reclaimers should not get too close to one another in terms of a safety distance
- The deliveries are prepared sequentially. However, suppose a reclaimer exhausts all cuts of a specific type of delivery. In that case, for idle reclaimer, we begin reclaiming the following delivery (with a different type of mineral) to save time and be more efficient. In our settings, we refer to this exception as *material exhaustion exception*.

Next, the idle reclaimer(s) select a job from its eligible neighbourhood as $e_{j,\phi_j} \in \mathcal{N}_t^r$. We evaluate the quality of each job as described in Section 7.1.1, and the selection is carried out according to the applied algorithm selection approach. This procedure is repeated for all reclaimers, and after completion of each job, the stockyard graph is updated. The reclaimers continue to reclaim minerals until we reach the specified tonnage of delivery d. The reclaimer will then begin reclaiming the next delivery $(d + 1)$. Note that if multiple reclaimers are idle at the same time, we evaluate a combination of jobs with respect to the reclaimers. This cycle is repeated for each delivery until all deliveries are completed.

Algorithm 7.1: Solution Construction Heuristic

Input : Selection algorithm from [DGA, RGA, MMAS]

$t \leftarrow 0$
$x := \emptyset$ ▷ Let x be the solution set
$\text{job}^r \leftarrow e_{c_0^r}, \phi_0 \quad \forall r \in R$ ▷ Add initial job to the queue
foreach $r \in R$ **do**
    $d^r \leftarrow 1$
    $\text{status}^r \leftarrow busy$
    $t^r \leftarrow T_{\text{job}^r}$ ▷ Record completion time
while $d \leq D$ ▷ all deliveries are not planned
do
    $t \leftarrow t + 1$ ▷ next time step
    foreach $r \in R$ **do**
        if $t^r \leq t$ **then**
            $x_d^r \leftarrow \text{job}^r$ ▷ Add completed job to x
            $\Gamma_{x_d} = \Gamma_{x_d} + \Gamma_{\text{job}^r}$ ▷ Tracking tonnage of the delivery
            if $\Gamma_{x_d} > \Gamma_d$ **then**
                $d = d + 1$ ▷ We start reclaiming the next delivery
            Generate $\mathcal{N}_t^r$ ▷ wrt. d^r
            while $N_t^r = \emptyset \wedge d^r <= D$ **do**
                $d^r = d^r + 1$ ▷ material exhaustion exception
                Generate $\mathcal{N}_t^r$
    $\mathcal{N}_t = \prod_r^R N_t^r \ \forall r \in R$ if r is idle and type_d^r is identical
    Evaluation of all candidate jobs in $\mathcal{N}_t$
    Selection of next job wrt. to input algorithm probability of selection

7.2.2 Deterministic And Randomised Greedy Algorithm

Similar to the initial study in Chapter 6, we use deterministic and randomised greedy algorithm as shown in Algorithm 7.2. The deterministic greedy algorithm (DGA) follows the principles introduced in the previous chapter where DGA selects the best eligible job in the neighbourhood of a reclaimer with the highest greediness at each time step. We also noticed that adding randomisation to DGA can result in finding better solutions. Recall that randomised greedy algorithm (RGA) has a greedy control parameter $\lambda \geq 0$. When λ is small, RGA gives weight to candidates in the selection where there is still a chance for weaker candidates to be selected.

The probability of selection of a cut in a neighbourhood of reclaimers at time step t for RGA is:

$$p(e_{j,\phi_j}|\mathcal{N}_t) = \frac{\eta(e_{j,\phi_j})^\lambda}{\sum_{e_{l,\phi_l} \in \mathcal{N}_t} \eta(e_{l,\phi_l})^\lambda}$$

This probabilistic selection is analogous to a roulette wheel, with the probability of selecting a candidate in $\mathcal{N}_t$ is proportionate to $\eta(e_{j,\phi_j})^\lambda$. η refers to the heuristic information, and we calculate it as follows.

In the previous chapter, we used some arbitrary numbers to determine the fitness proportionate based selection. However, in this chapter, we use the following principle. If the average target quality and window target quality for all candidates are zero (

Algorithm 7.2: Greedy Algorithms

$x := \emptyset$
Construct a solution x step by step, using Algorithm 7.1 with following selection probability
if *Selection is deterministic (DGA)* **then**
    Choose a job $e^* = \text{argmin}_{e_{j,\phi_j} \in \mathcal{N}_t} f(e_{j,\phi_j})$
else
    ▷ RGA
    Choose a job (e^*) according to probability
    $$p(e_{j,\phi_j}|\mathcal{N}_t) = \frac{\eta(e_{j,\phi_j})^\lambda}{\sum_{e_{l,\phi_l} \in \mathcal{N}_t} \eta(e_{l,\phi_l})^\lambda}$$

$v_1(e_{j,\phi_j}) = 0 \wedge v_2(e_{j,\phi_j}) = 0 \quad \forall e_{j,\phi_j} \in \mathcal{N}_t)$, $\eta(e_{j,\phi_j})$ will be as identical as $u(e_{j,\phi_j})$. Otherwise, we employ linear ranking as follows. We sort the jobs from the best to the worst with index i from 0 to $|\mathcal{N}_t| - 1$. We calculate the probability of selection based on their rank as follows [ES15].

$$\eta_{e_{i,\phi_i}} = \frac{2 - SP}{\mu} + \frac{2i(SP - 1)}{\mu(\mu - 1)}$$

Figure 7.2 shows the behaviour of the preceding equation with respect to different values of λ and $1 < SP \leq 2$ where the latter refers to the selection pressure parameter. We assume that there are 10 jobs available in the neighbourhood of the reclaimer to show the behaviour of the probability of selection. According to the fitness function, these jobs are sorted from worst to the best. We can see that when λ and SP are small, the selection probability of jobs is very close to each other. However, adding λ to increase the RGA greediness and SP for the selection pressure results in more than 80% chance of selection for the best job.

7.2.3 Max-Min Ant System (MMAS)

MMAS in this chapter also follows the same principles introduced before in Section 6 and Section 3.9. We focus mainly on the MMAS parameters here where MMAS generates n artificial ants for its colony at the first generation. Each ant performs a random walk using the solution construction heuristic to generate a valid solution step by step. employing the following probabilistic selection strategy for our problem with multiple reclaimers we have

$$p(e_{j,\phi_j}|\mathcal{N}_t) = \frac{[\xi_{e_{j,\phi_j}}]^\alpha [\eta(e_{j,\phi_j})]^\beta}{\sum_{e_{l,\phi_l} \in \mathcal{N}_t} [\xi_{e_{l,\phi_l}}]^\alpha [\eta(e_{l,\phi_l})]^\beta}$$

where α and β are MMAS parameters that regulate the influence of heuristic and pheromone information, respectively, the preceding equation implies that the selection of the next cut is both dependent on the quality of jobs in terms of the objective function and the pheromone information.

After all ants complete their random walk, pheromone evaporation occurs where it aids in avoiding the unvisited paths as follows.

$$\xi_{e_{j,\phi_j}} = (1 - \rho).\xi_{e_{j,\phi_j}}$$

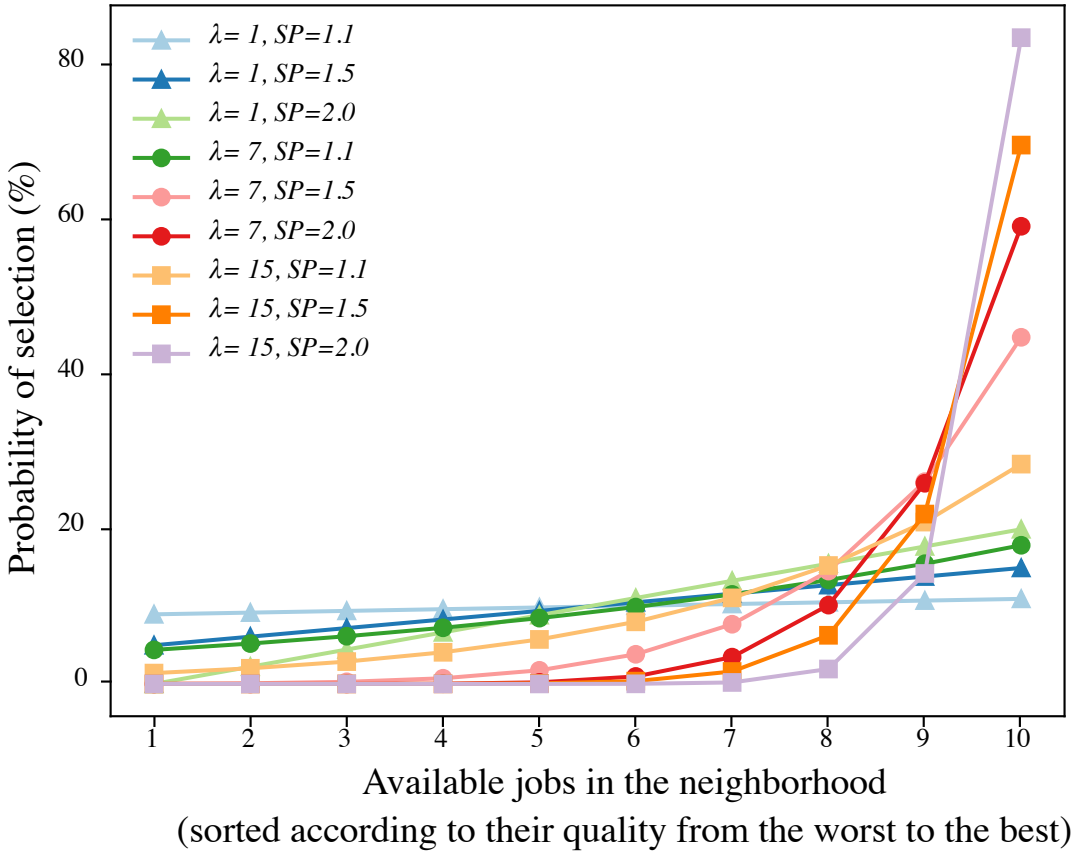

FIGURE 7.2. Probability of selection when linear ranking is active for different values of λ and SP

where $0 < \rho \leq 1$ denotes the evaporation rate. Then, one specified ant (π^*) deposits pheromone on their solution (x^*) edges as follows.

$$\xi_{i,j,\phi_i,\phi_j} = \begin{cases} \min\{\xi_{e_{j,\phi_j}} + \rho, 1 - \frac{1}{J}\} & \text{if } (e_{j,\phi_j}) \in x^* \\ \max\{\xi_{e_{j,\phi_j}}, \frac{1}{J}\} & \text{otherwise.} \end{cases}$$

π^* could be the best-so-far-ant (BSFA) or the iteration-best-ant (IBA).

7.2.4 Iterative Local Search

In the previous chapter, we applied iterative local search (ILS) to improve the quality of solutions obtained by MMAS. However, for a more realistic setting of stockyard recovery problem with interaction of multiple reclaimers in delivering sequentially, it is hard to define a local search neighbourhood for an obtained schedule produced for multiple reclaimers which each delivery is scheduled sequentially. The precedence connection between different reclaimers with respect to shared rows can complicate the issue. As a result, a small neighbourhood is defined as follows. We consider the deliveries in order and we begin with the first reclaimer, as $x_{d=1}^{r=1}$.

In this segment, we can swap the positions of two adjacent jobs to get the perturbed solution as x^*. Next, we validate the precedence constraints with respect to all reclaimers. If it meets the constraints, we evaluate the change in the objective function. Note that swapping adjacent jobs in a the defined segment only affects $v_2(x)$

Algorithm 7.3: Iterative Local Search for MMAS

```
Input: x = Initial Solution
Stop criterion ← False
while Stop criterion = False do
    foreach d ∈ D do
        foreach r ∈ R do
            Successful swap ← ∅
            foreach e_{j,φ_j} ∈ x_d^r do
                x* = Swap(x)        ▷ Swap this job with its succeeding adjacent
                if precedence constraints in x* are met then
                    Calculate new completion time for affected jobs
                    Calculate u(x*) and v_2(x*)
                    if f(x*) < f(x) then
                        Record x* as a successful swap
            x ← Best solution out of recorded solutions in successful swaps
return x
```

and $u(x)$, but $v_1(x)$ remains unchanged. Due to this neighbourhood, $v_1(x)$ does not change and only $v_2(x)$ and $u(x)$ are affected. If the perturbed solution outperforms the original solution, we mark it as a successful swap, and if necessary, we reorder the jobs to ensure time consistency in terms of affected jobs' completion time.

After swapping all adjacent jobs in $x_{d=1}^{r=1}$, the best one from the recorded ones replaces the original solution, and we repeat the local search on the same segment. The process is repeated until no improvement is observed. Then we proceed to the next segment as the next reclaimer for the same delivery, and the iterative local search technique described above is repeated.

7.3 Experimental Setup

In this section, we detail our experimental setup and assumptions and fine-tune the algorithms we described in the previous section.

7.3.1 Problem Setup

We consider a real-world discretised stockyard model provided by our industrial partner shown in Fig 7.1 that described previously. We suppose that no more than ten deliveries should be planned (according to available material in the stockpile), with each delivery requiring a tonnage in range of [100,000, 200,000] tonnes. Note that the stockyard dataset allows us to schedule up to 10 deliveries, whereas more than 9 deliveries can be processed only with 3 reclaimers.

We define an instance as a three-tuple (R, D, ϕ) with the following components in order: number of deliveries, number of reclaimers and how many reclaiming direction is possible. For example, (6-2-2) depicts a situation in which six deliveries should be planned, two reclaimers are available, and reclaimers can employ both ϕ_1 and ϕ_2 reclaiming directions. In all instances, we evaluate the objective function in the form $(v_1(x), v_2(x), u(x))$.

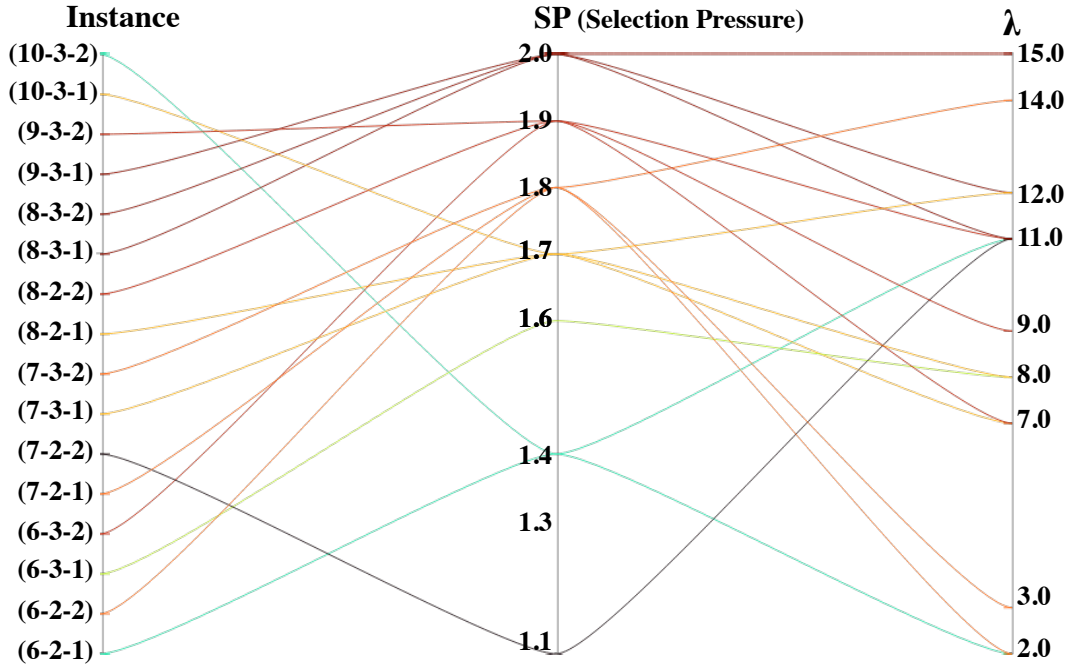

FIGURE 7.3. Best parameter configurations for RGA

7.3.2 Algorithm Setup

We see that our algorithms have several parameters that can influence the computational cost and their behaviour to discover good solutions.

DGA is deterministic, and there is no configuration parameter. On the other hand, RGA has two parameters λ and SP to adjust greediness and selection pressure, respectively. We use $\lambda \in \{1, 2, 3, \ldots, 15\}$ and $SP \in \{1.1, 1.2, 1.3, ..., 2\}$.

For MMAS, we set the number of ants as 10 and the termination criterion as 1000 maximum generations since they affect the computational expense. MMAS has parameters of α, β, ρ, n, and π^* to be configured as follows.

- $\alpha \in \{1, 2, 3 \ldots, 10\}$
- $\beta \in \{1, 2, 3 \ldots, 10\}$
- $\rho \in \{0.1, 0.2, 0.3, \ldots, 1.0\}$
- $SP \in \{1.1, 1.2, 1.3, \ldots, 2.0\}$
- $\pi^* \in \{BSFA, IBA\}$.

To find a suitable parameter configuration, we use the Irace software package [Lóp+16] which employs the method of 1/F-Race [Bir+10] for automatic algorithm configuration. We use the default parameters and limit the number of experiments to 5000 as the termination criterion for parameter tuning.

Figure 7.3 and 7.4 show the configurations obtained by Irace for each instance for RGA and MMAS, respectively. These parallel coordinate plots illustrate the value of each parameter for an obtained setup which is represented by a line. Note that these plots only show the configuration obtained for instances with more than 6 deliveries.

For RGA, we can see that selection pressures greater than 1.7 are more favourable; the greedy parameter value should be small or large, with no value in the range of 4-6.

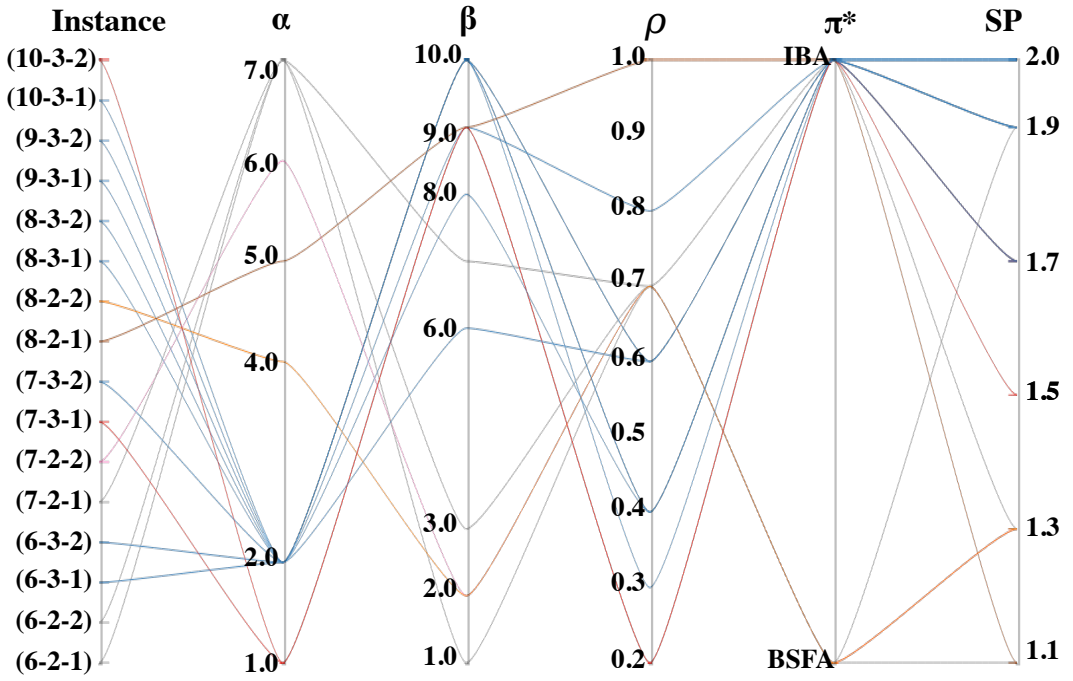

FIGURE 7.4. Best parameter configurations for MMAS

10 and 11 have not been identified for tuning. For MMAS, we can see that selection pressure follows the same pattern in RGA. IBA is the best choice for updating the pheromones for most instances. All values other than 0.1, 0.5, and 0.9 have been utilised for evaporation rate (ρ). We can also see that the chosen values for α and β are substantially different from ACO's most typical settings of $\alpha = 1$ and $\beta = 2$.

The source code has been implemented in Python and can be found in `https://git.io/JyUI1`. This framework includes the algorithms mentioned before and the simulation tool to replicate the experiments. The framework also includes instructions on how to run the code.

7.4 Results

We run RGA and MMAS on each instance 51 times to obtain reliable results and the median could be easily calculated. We use the Kruskal-Wallis test with a 95% confidence interval to compare randomised algorithms and check if there is a significant difference between them. Next, for pair-wise comparisons, we use the Bonferroni posteriori approach to correct the p-values. We rank the obtained solutions according to our lexicographic objective function, and we use this ranking to perform the statistical comparison.

Tables 7.1- 7.4 list the objective function obtained for optimised solution for planning deliveries with more than 2 deliveries. In reported tables, we list the median, best and worst solutions obtained for our randomised algorithms. We report the same value for DGA because it is deterministic.

We use a success rate (SR) indicator to show the percentage of observations for a randomised algorithm where the first component of the objective function (the most important objective) for an obtained solution is zero $v_1(x) = 0$. Note that SR for DGA (as a deterministic algorithm) is either 0.0 or 100.0.

TABLE 7.1. Objective functions obtained for the solutions in for instances with 2-3 deliveries

Instance		DGA	RGA	MMAS	MMAS-local
(2-2-1)	Median	(0.0, 0.0, 63.1384)	(0.0, 0.0092, 64.8688)	**(0.0, 0.0, 49.6593)**	(0.0, 0.0, 49.9543)
	Best	(0.0, 0.0, 63.1384)	(0.0, 0.0, 54.2741)	**(0.0, 0.0, 47.6128)**	**(0.0, 0.0, 47.6128)**
	Worst	(0.0, 0.0, 63.1384)	(0.0, 0.1532, 77.3449)	(0.0, 0.0, 51.3663)	**(0.0, 0.0, 51.3335)**
	SR	100.0	100.0	100.0	100.0
(2-2-2)	Median	(0.0, 0.0, 61.8026)	(0.0, 0.0, 70.7213)	**(0.0, 0.0, 48.9521)**	(0.0, 0.0, 49.2941)
	Best	(0.0, 0.0, 61.8026)	(0.0, 0.0, 55.3678)	**(0.0, 0.0, 48.3642)**	(0.0, 0.0, 48.3981)
	Worst	(0.0, 0.0, 61.8026)	(0.0, 0.1018, 60.336)	**(0.0, 0.0, 50.6601)**	(0.0, 0.0, 51.6192)
	SR	100.0	100.0	100.0	100.0
(2-3-1)	Median	(0.0, 0.0, 57.083)	(0.0, 0.045, 69.4875)	**(0.0, 0.0, 47.9265)**	(0.0, 0.0, 48.2739)
	Best	(0.0, 0.0, 57.083)	(0.0, 0.0, 53.3086)	(0.0, 0.0, 47.2892)	**(0.0, 0.0, 45.698)**
	Worst	(0.0, 0.0, 57.083)	(0.0, 0.0, 76.5126)	(0.0, 0.0, 49.4229)	**(0.0, 0.0, 48.9782)**
	SR	100.0	98.0	100.0	100.0
(2-3-2)	Median	(0.0, 0.0, 57.083)	(0.0, 0.0, 69.0928)	**(0.0, 0.0, 47.6185)**	(0.0, 0.0, 48.2093)
	Best	(0.0, 0.0, 57.083)	(0.0, 0.0, 58.0832)	(0.0, 0.0, 47.235)	**(0.0, 0.0, 47.1949)**
	Worst	(0.0, 0.0, 57.083)	(0.0, 0.2135, 71.215)	**(0.0, 0.0, 49.4745)**	(0.0, 0.0, 49.8916)
	SR	100.0	100.0	100.0	100.0
(3-2-1)	Median	(0.0, 0.0575, 92.4521)	(0.0, 0.0092, 90.2826)	(0.0, 0.0, 65.9564)	**(0.0, 0.0, 63.146)**
	Best	(0.0, 0.0575, 92.4521)	(0.0, 0.0, 74.4169)	(0.0, 0.0, 61.6592)	**(0.0, 0.0, 61.6415)**
	Worst	(0.0, 0.0575, 92.4521)	(0.0, 0.2174, 89.6102)	(0.0, 0.0, 68.832)	**(0.0, 0.0, 67.9776)**
	SR	100.0	100.0	100.0	100.0
(3-2-2)	Median	(0.0, 0.0, 89.3236)	(0.0, 0.0, 86.6843)	(0.0, 1.3757, 239.996)	**(0.0, 0.1608, 214.029)**
	Best	(0.0, 0.0, 89.3236)	(0.0, 0.0, 78.28)	(0.0, 0.8926, 228.8528)	**(0.0, 0.055, 205.8034)**
	Worst	(0.0, 0.0, 89.3236)	**(0.0, 0.1336, 104.1021)**	(0.0003, 3.9321, 222.0364)	(0.0059, 1.3283, 192.701)
	SR	100.0	100.0	96.0	88.0
(3-3-1)	Median	(0.0, 0.0, 75.4511)	(0.0, 0.0202, 77.6254)	**(0.0, 0.0, 59.447)**	(0.0, 0.0, 64.383)
	Best	(0.0, 0.0, 75.4511)	(0.0, 0.0, 68.3009)	(0.0, 0.0, 59.2653)	**(0.0, 0.0, 59.0969)**
	Worst	(0.0, 0.0, 75.4511)	(0.0, 0.2248, 73.6197)	**(0.0, 0.0, 64.0279)**	(0.0, 0.0, 65.028)
	SR	100.0	100.0	100.0	100.0
(3-3-2)	Median	(0.0, 0.0, 75.1823)	(0.0, 0.1898, 84.047)	**(0.0, 0.0, 61.9133)**	(0.0, 0.0, 63.1242)
	Best	(0.0, 0.0, 75.1823)	(0.0, 0.0, 72.6353)	(0.0, 0.0, 59.2794)	**(0.0, 0.0, 59.0633)**
	Worst	(0.0, 0.0, 75.1823)	(0.0, 0.1998, 97.1761)	**(0.0, 0.0, 67.141)**	(0.0, 0.0, 69.3126)
	SR	100.0	100.0	100.0	100.0

TABLE 7.2. Objective functions obtained for the solutions in for instances with 4-5 deliveries

Instance		DGA	RGA	MMAS	MMAS-local
(4-2-1)	Median	(0.0, 58.5812, 142.4239)	(0.0, 57.9318, 130.3273)	**(0.0, 54.2947, 145.6548)**	(0.0, 54.7247, 142.0836)
	Best	(0.0, 58.5812, 142.4239)	(0.0, 57.6087, 127.747)	(0.0, 53.981, 147.888)	**(0.0, 53.9013, 139.6754)**
	Worst	(0.0, 58.5812, 142.4239)	(0.0, 60.3444, 136.7928)	(0.0, 54.8605, 148.6741)	**(0.0, 54.7413, 158.896)**
	SR	100.0	98.0	100.0	100.0
(4-2-2)	Median	(0.0063, 60.835, 144.9296)	(0.0, 57.9492, 132.1126)	(0.0, 54.0759, 157.611)	**(0.0, 53.8887, 181.8816)**
	Best	(0.0063, 60.835, 144.9296)	(0.0, 57.6182, 139.6121)	(0.0, 53.9911, 168.2198)	**(0.0, 53.8226, 154.0985)**
	Worst	(0.0063, 60.835, 144.9296)	(0.0, 60.9878, 141.42)	(0.0, 54.2289, 184.7501)	**(0.0, 54.0178, 168.1702)**
	SR	0.0	98.0	100.0	100.0
(4-3-1)	Median	(0.0, 67.5545, 146.0696)	(0.0, 66.7489, 142.2)	**(0.0, 57.4258, 149.6348)**	(0.0, 57.4523, 133.0181)
	Best	(0.0, 67.5545, 146.0696)	(0.0, 61.5783, 174.9475)	(0.0, 56.9379, 149.3044)	**(0.0, 56.7025, 127.3126)**
	Worst	(0.0, 67.5545, 146.0696)	(0.0, 70.9181, 158.1506)	**(0.0, 57.639, 159.8935)**	(0.0, 57.6516, 147.3138)
	SR	100.0	100.0	100.0	100.0
(4-3-2)	Median	(0.0, 71.9556, 157.8473)	(0.0, 70.973, 169.9409)	(0.0, 57.4998, 186.9824)	**(0.0, 57.4431, 170.5191)**
	Best	(0.0, 71.9556, 157.8473)	(0.0, 63.7771, 155.4851)	(0.0, 56.8679, 144.0387)	**(0.0, 56.4915, 163.4503)**
	Worst	(0.0, 71.9556, 157.8473)	(0.0, 75.4247, 176.1717)	(0.0, 58.8066, 172.2938)	**(0.0, 58.4295, 164.8766)**
	SR	100.0	100.0	100.0	100.0
(5-2-1)	Median	(0.0438, 102.1746, 173.9782)	(0.0273, 101.718, 182.0756)	(0.0012, 102.6828, 228.5581)	**(0.0011, 99.5477, 224.1774)**
	Best	(0.0438, 102.1746, 173.9782)	(0.0046, 103.263, 167.8441)	(0.001, 100.9133, 229.5818)	**(0.001, 98.2025, 239.3864)**
	Worst	(0.0438, 102.1746, 173.9782)	(0.1935, 103.6282, 243.1028)	**(0.0016, 103.5916, 223.3174)**	(0.0018, 99.4701, 210.7622)
	SR	0.0	0.0	0.0	0.0
(5-2-2)	Median	(0.0235, 103.1665, 175.0023)	(0.0131, 101.6707, 181.8164)	**(0.0009, 102.9038, 302.4058)**	(0.001, 99.0435, 312.2554)
	Best	(0.0235, 103.1665, 175.0023)	(0.0078, 102.138, 173.9245)	(0.0009, 102.1923, 328.4197)	**(0.0009, 98.9954, 259.8169)**
	Worst	(0.0235, 103.1665, 175.0023)	(0.1824, 105.088, 233.3203)	**(0.001, 107.2824, 254.6951)**	(0.0011, 102.7451, 295.7914)
	SR	0.0	0.0	0.0	0.0
(5-3-1)	Median	(0.0, 105.3406, 173.8627)	(0.0, 105.8641, 168.4634)	**(0.0, 95.578, 155.9611)**	(0.0, 96.8763, 164.1394)
	Best	(0.0, 105.3406, 173.8627)	(0.0, 99.5648, 172.6111)	**(0.0, 95.1011, 152.7925)**	(0.0, 95.1105, 170.8295)
	Worst	(0.0, 105.3406, 173.8627)	(0.0, 107.851, 188.782)	(0.0, 97.2515, 160.8189)	**(0.0, 97.1006, 169.9686)**
	SR	100.0	100.0	100.0	100.0
(5-3-2)	Median	(0.0, 106.2677, 181.9484)	(0.0, 108.1915, 206.6663)	(0.0, 99.752, 184.9368)	**(0.0, 97.9266, 174.3664)**
	Best	(0.0, 106.2677, 181.9484)	(0.0, 103.0152, 194.1108)	(0.0, 97.9713, 182.2523)	**(0.0, 96.924, 159.4967)**
	Worst	(0.0, 106.2677, 181.9484)	(0.0, 113.0368, 210.7794)	**(0.0, 101.0779, 196.4892)**	(0.0, 101.6432, 193.4623)
	SR	100.0	100.0	100.0	100.0

TABLE 7.3. Objective functions obtained for the solutions in for instances with 6-7 deliveries

Instance		DGA	RGA	MMAS	MMAS-local
(6-2-1)	Median	(0.0438, 102.1746, 197.4267)	(0.0046, 104.2092, 246.6884)	(0.0011, 105.0168, 319.2173)	**(0.001, 99.2849, 313.7267)**
	Best	(0.0438, 102.1746, 197.4267)	(0.0046, 104.2092, 246.6884)	(0.0009, 106.1539, 313.9708)	**(0.0009, 99.832, 297.235)**
	Worst	(0.0438, 102.1746, 197.4267)	(0.3946, 115.1596, 314.2067)	**(0.0013, 104.9209, 319.558)**	(0.0014, 100.8427, 300.9663)
	SR (%)	0.0	0.0	0.0	0.0
(6-2-2)	Median	(0.0235, 103.1665, 220.0882)	(0.0153, 103.351, 303.5167)	(0.001, 102.6809, 407.1123)	**(0.0009, 101.173, 366.9152)**
	Best	(0.0235, 103.1665, 220.0882)	(0.0024, 102.9986, 273.2049)	(0.0009, 101.834, 429.2643)	**(0.0009, 98.2865, 369.7729)**
	Worst	(0.0235, 103.1665, 220.0882)	(0.119, 104.6899, 371.9484)	(0.001, 105.9105, 412.3481)	**(0.001, 100.9826, 388.3912)**
	SR (%)	0.0	0.0	0.0	0.0
(6-3-1)	Median	(0.0, 105.3406, 215.4017)	(0.0, 107.8442, 222.0722)	(0.0, 98.1578, 199.2548)	**(0.0, 96.0026, 196.5737)**
	Best	(0.0, 105.3406, 215.4017)	(0.0, 101.5879, 216.1834)	(0.0, 96.1019, 205.8208)	**(0.0, 95.2302, 192.9204)**
	Worst	(0.0, 105.3406, 215.4017)	(0.0, 110.0313, 235.1967)	(0.0, 98.2292, 195.2234)	**(0.0, 97.9712, 183.2543)**
	SR (%)	100.0	100.0	100.0	100.0
(6-3-2)	Median	(0.0, 107.1645, 224.3089)	(0.0, 108.7695, 237.8931)	(0.0, 99.8783, 195.6318)	**(0.0, 99.6923, 205.196)**
	Best	(0.0, 107.1645, 224.3089)	(0.0, 103.2542, 223.0722)	**(0.0, 97.6263, 255.3897)**	(0.0, 97.7767, 192.1045)
	Worst	(0.0, 107.1645, 224.3089)	(0.0, 112.1085, 252.1892)	(0.0, 100.622, 219.9266)	**(0.0, 100.5491, 213.4196)**
	SR (%)	100.0	100.0	100.0	100.0
(7-2-1)	Median	(0.0438, 102.1746, 216.317)	(0.0462, 102.5059, 234.9085)	(0.0011, 103.0838, 331.9113)	**(0.0011, 100.5098, 346.8453)**
	Best	(0.0438, 102.1746, 216.317)	(0.0106, 100.4892, 227.1779)	**(0.0009, 102.0894, 341.0696)**	(0.001, 98.3401, 330.7355)
	Worst	(0.0438, 102.1746, 216.317)	(0.2418, 113.3867, 321.4938)	**(0.0012, 103.8423, 350.8519)**	(0.0013, 99.0285, 323.713)
	SR (%)	0.0	0.0	0.0	0.0
(7-2-2)	Median	(0.0235, 103.1665, 238.108)	(0.0352, 104.8558, 310.5028)	(0.0009, 107.5879, 437.6702)	**(0.0009, 100.6736, 394.1645)**
	Best	(0.0235, 103.1665, 238.108)	(0.0024, 103.6803, 323.8809)	(0.0009, 103.1755, 417.8908)	**(0.0009, 98.3221, 390.2379)**
	Worst	(0.0235, 103.1665, 238.108)	(0.1739, 105.3239, 392.1911)	(0.001, 106.6327, 419.3385)	**(0.001, 100.6576, 398.1106)**
	SR (%)	0.0	0.0	0.0	0.0
(7-3-1)	Median	(0.0, 105.3406, 224.2254)	(0.0026, 105.2915, 246.4175)	(0.0, 98.2164, 222.4849)	**(0.0, 98.0351, 208.0818)**
	Best	(0.0, 105.3406, 224.2254)	(0.0, 100.399, 282.8226)	(0.0, 95.3277, 210.943)	**(0.0, 95.2497, 209.0721)**
	Worst	(0.0, 105.3406, 224.2254)	(0.0026, 105.2915, 246.4175)	(0.0, 99.8691, 230.628)	**(0.0, 99.6428, 214.9185)**
	SR (%)	100.0	98.0	100.0	100.0
(7-3-2)	Median	(0.0, 107.1645, 233.1917)	(0.0, 108.2167, 266.023)	(0.0, 99.8828, 234.1908)	**(0.0, 98.954, 279.1073)**
	Best	(0.0, 107.1645, 233.1917)	(0.0, 103.449, 262.2946)	(0.0, 97.9256, 211.2687)	**(0.0, 97.1307, 266.3458)**
	Worst	(0.0, 107.1645, 233.1917)	(0.0005, 106.7591, 259.0931)	**(0.0, 100.7634, 209.6184)**	(0.0, 101.4503, 220.1084)
	SR (%)	100.0	97.0	100.0	100.0

TABLE 7.4. Objective functions obtained for the solutions in for instances with 8-10 deliveries

Instance		DGA	RGA	MMAS	MMAS-local
(8-2-1)	Median	(0.0438, 102.1746, 251.1864)	(0.0273, 101.6469, 265.3198)	(0.001, 107.6261, 340.2463)	**(0.001, 100.4274, 382.1242)**
	Best	(0.0438, 102.1746, 251.1864)	(0.0061, 99.7059, 269.7227)	(0.001, 104.662, 364.4957)	**(0.001, 99.411, 366.9006)**
	Worst	(0.0438, 102.1746, 251.1864)	(0.3213, 106.2929, 350.9886)	**(0.0014, 108.9955, 350.0817)**	(0.0015, 101.8406, 334.152)
	SR (%)	0.0	0.0	0.0	0.0
(8-2-2)	Median	(0.0235, 103.2428, 263.9633)	(0.051, 102.3833, 279.9597)	(0.001, 103.8465, 438.6782)	**(0.0009, 98.6669, 419.5654)**
	Best	(0.0235, 103.2428, 263.9633)	(0.0099, 100.5136, 281.1246)	(0.0009, 104.757, 464.8563)	**(0.0009, 98.4481, 420.6694)**
	Worst	(0.0235, 103.2428, 263.9633)	(0.2667, 104.7104, 399.3313)	(0.0011, 110.8108, 428.5091)	**(0.001, 101.8508, 410.8488)**
	SR (%)	0.0	0.0	0.0	0.0
(8-3-1)	Median	(0.0, 105.3626, 261.8697)	(0.0, 104.4384, 311.0165)	**(0.0, 97.9676, 285.1839)**	(0.0, 97.9992, 248.4489)
	Best	(0.0, 105.3626, 261.8697)	(0.0, 100.5335, 308.9175)	(0.0, 96.2408, 234.5436)	**(0.0, 96.2062, 257.1488)**
	Worst	(0.0, 105.3626, 261.8697)	(0.0014, 105.9245, 270.8747)	**(0.0, 98.1712, 308.6494)**	(0.0, 99.0334, 242.9122)
	SR (%)	100.0	92.0	100.0	100.0
(8-3-2)	Median	(0.0, 107.1645, 270.9)	(0.0, 103.0842, 293.0588)	**(0.0, 98.9443, 259.0835)**	(0.0, 99.6151, 237.5469)
	Best	(0.0, 107.1645, 270.9)	(0.0, 103.0842, 293.0588)	(0.0, 98.1982, 281.7398)	**(0.0, 97.8378, 263.3097)**
	Worst	(0.0, 107.1645, 270.9)	(0.0026, 108.1778, 299.4416)	**(0.0, 100.4589, 293.1712)**	(0.0, 100.4663, 267.3126)
	SR (%)	100.0	91.0	100.0	100.0
(9-3-1)	Median	(0.0, 105.4089, 302.9124)	(0.0, 106.8992, 287.1712)	(0.0, 97.7961, 281.3341)	**(0.0, 97.7619, 285.255)**
	Best	(0.0, 105.4089, 302.9124)	(0.0, 100.1781, 331.6884)	(0.0, 96.3332, 292.1141)	**(0.0, 96.0063, 274.6188)**
	Worst	(0.0, 105.4089, 302.9124)	(0.0131, 103.7835, 356.1299)	(0.0, 98.7486, 270.1143)	**(0.0, 98.033, 276.9183)**
	SR (%)	100.0	85.0	100.0	100.0
(9-3-2)	Median	(0.0, 107.1828, 321.2739)	(0.0, 105.8886, 345.9241)	**(0.0, 98.8647, 305.6628)**	(0.0, 100.4717, 305.733)
	Best	(0.0, 107.1828, 321.2739)	(0.0, 105.1871, 342.7084)	(0.0, 98.8647, 305.6628)	**(0.0, 98.6388, 289.2822)**
	Worst	(0.0, 107.1828, 321.2739)	(0.0035, 107.1188, 353.2033)	(0.0, 101.6301, 324.3876)	**(0.0, 100.6362, 322.0756)**
	SR (%)	100.0	94.0	100.0	100.0
(10-3-1)	Median	(0.0, 105.4252, 311.0819)	(0.045, 100.8182, 367.499)	**(0.0, 97.4539, 303.5063)**	(0.0, 98.0039, 291.9887)
	Best	(0.0, 105.4252, 311.0819)	(0.0, 102.7394, 311.4885)	(0.0, 96.3966, 300.2111)	**(0.0, 96.1314, 347.2854)**
	Worst	(0.0, 105.4252, 311.0819)	(0.045, 100.8182, 367.499)	**(0.0, 98.3585, 309.6789)**	(0.0, 98.5017, 309.4715)
	SR (%)	100.0	72.0	100.0	100.0
(10-3-2)	Median	(0.0, 107.1865, 339.3049)	(0.0, 109.506, 387.0197)	(0.0, 100.4038, 320.7127)	**(0.0, 99.2589, 327.916)**
	Best	(0.0, 107.1865, 339.3049)	(0.0, 104.5479, 358.7583)	**(0.0, 98.3857, 319.6994)**	(0.0, 98.8018, 322.0849)
	Worst	(0.0, 107.1865, 339.3049)	(0.0114, 110.5372, 390.1529)	(0.0, 102.8742, 319.4772)	**(0.0, 101.8307, 325.0809)**
	SR (%)	100.0	82.0	100.0	100.0

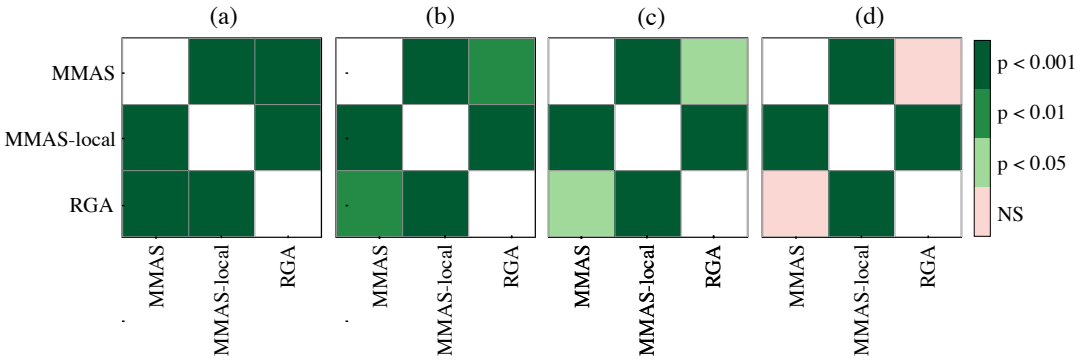

FIGURE 7.5. Significance plot of statistical tests for randomised algorithms for different instances. p refers to the p-value and NS shows no significant difference. Each subplot refers to different instances as follows. (a): (3-2-2), (4-2-1), (4-2-2), (4-3-2), (5-3-2), (7-2-2), (9-3-2). (b): (5-2-1), (7-3-2). (c): (6-3-2), (d): other instances.

Figure 7.5 shows the statistical significance for all instances. There are four categories of significant differences observed. In all cases, there is a strongly significant difference between MMAS-local and other algorithms. In some instances, tuned MMAS and RGA are weakly different; however, they are not significantly different in most instances.

DGA is successful in finding solution with 100% SR in 10 instances out of 16 where number of deliveries are more than six. As observed in Chapter 6, DGA exhaust good material early in the stages of planning. In all instances, RGA outperforms DGA. We can see the same trend in outperforming RGA by MMAS.

We can see that planning six deliveries or less using two reclaimers instead of three can lead to a slight violation in $v_1(x)$. However, planning by three reclaimers results in a 100% success rate for all randomised algorithms except in instance of (2-3-1) that we see 98% success rate for RGA.

For two deliveries which is the most straightforward problem, we can see that DGA is weaker than MMAS. MMAS-local and MMAS find the best solution. However, when adding the second reclaimer direction, MMAS can outperform the MMAS-local. Scheduling two deliveries using three reclaimers leads to lower utility values than scheduling with two reclaimers. Adding a delivery into the schedule to provide three deliveries using two reclaimers makes DGA find a solution with violated target window quality. However, MMAS-local finds the best solution with a 100% success rate and no target quality violation. Similar to two delivery instances, we can see that adding three reclaimers results in providing the delivery faster. In addition, using two reclaiming directions leads to target quality violation; however, using three reclaimers tackles this issue.

Four deliveries planning makes the problem more complex where all the obtained solutions violate the window target quality objective. However, for all directions and the number of reclaimers, MMAS-local can find the best solution compared to other methods. We can see that adding reclaimers can reduce the delivery time, but it also exacerbates the window target quality. For example, comparing instances (4-3-1) with (4-2-1), the former best solution has found a solution with better utility but a worse window target quality.

Five deliveries scheduling can result in a violation in average target quality using two reclaimers, but this violation is eradicated when three reclaimers are used. As the number of delivery surpasses 6, we can see that RGA becomes less capable of

finding solutions with 100% SR. On the other hand, MMAS and MMAS-local can find solutions with 100% SR always for these instances. MMAS-local can outperform MMAS in most cases, but there exist some exceptions in our observations.

We see that for instance (6-3-2), the best solution obtained by MMAS is slightly better than the one obtained by MMAS-local with respect to $v_1(x)$. However, MMAS-local has found a worse solution, but with a better utility. The median and worst of obtained solutions by MMAS-local is better than corresponding ones obtained by MMAS. We can see the same trend for instances (7-2-1), (10-3-2). There could be different reasons for this limitation. It could be due to the nature of our local search, in which we just shift the jobs in a solution without changing their reclamation directions. Therefore, there is potential to expand on the local search strategy to consider the reclamation direction in specific where bi-directional reclamation is occurring such as instance (6-3-2).

Another possibility is that there is a trade-off between $v_1(x)$ and $u(x)$. For instance (7-2-1), the difference in $v_1(x)$ between the solutions generated by MMAS-local and MMAS is 0.0001 for the best obtained solution. We can see that MMAS-local has obtained a better solution despite the minor variance in the first component. According to this observation, it is better to define a threshold to have flexibility in dealing with hard constraints on comparing two solutions to identify more practical solutions.

Using two reclaimers for 7-8 delivery scheduling leads the algorithm to find solutions with 0% success. Adding the third reclaimer to scheduling makes MMAS and MMAS-local 100% successful in finding solutions with no average target quality violation. For the 9-10 delivery schedule, we see that MMAS and its variant with local search can obtain solutions with a 100% success rate. However, using reclaimers with two directions for nine deliveries leads to findings solutions with worse window target quality and utility. However, the best solution obtained has a better utility for ten delivery but worse window target quality.

7.5 Conclusions

Stockpiles are essential in the mining value chain, assisting in maximising value and production. Quality control of taken minerals from the stockpiles is a significant concern for stockpile managers where failure to meet some requirements can lead to losing money. The previous chapter investigated the problem using a single reclaimer and basic assumptions. This chapter extended the approach to consider multiple reclaimers in preparing for short and long-term deliveries. The engagement of multiple reclaimers complicates the problem in terms of their interaction in preparing a delivery simultaneously and safety distancing of reclaimers. We also considered more realistic settings, such as handling different minerals with different types of reclaimers. We investigated deterministic and randomised greedy algorithms and a variation of ant colony optimisation. We also used the automatic parameter tuning method to fine-tune our algorithms and determine the best configuration for each instance to achieve better results. We also designed a local search operator for MMAS to more finely investigate a solution, and it can outperform other methods in most instances. Further research may include (1) adding a human in the design loop to revise the algorithms' solutions to facilitate the practical use of this work, (2) improving the local search to account for changes in the reclamation direction and adding a threshold for trade-off consideration among objectives, and (3) applying the platform to a dynamic problem of stacking and reclaiming a stockpile where in practice stacking occurs in parallel to reclaiming.

Chapter 8

Conclusions

Optimisation algorithms can either be designed for a specific problem or could be general-purpose. Bio-inspired optimisation methods are general-purpose optimisation techniques that simulate natural phenomena. This thesis aimed to examine bio-inspired methods to solve a range of combinatorial problems with different attributes across the spectrum from fundamental to real-world problems. These attributes include dynamic and stochastic problems, problems with different types of decision variables in a solution representation and constructing a solution for scheduling problems step by step.

We began Chapter 2 by outlining the combinatorial problems covered in the subsequent chapters. After that, we discussed the deterministic and randomised methods used to solve the optimisation problems in Chapter 3. Our first investigation step was to solve the well-known knapsack problem with dynamic changes in the capacity over time and stochastic weights of items in Chapter 4. We applied chance constraint programming to convert the deterministic knapsack constraint to the probabilistic constraint, where the probability of violating the capacity should be lower than a threshold. We estimated the probabilistic term using inequality tails, namely Chebyshev's inequality and Chernoff bound. We integrated these inequality tails with baseline evolutionary algorithms in single and bi-objective settings. In the bi-objective setting, we introduced new helper objective functions. Our results showed that our approach leads to more efficient solutions in comparison with the single-objective approach. Moreover, our approach using its population can adapt to the dynamic environment of our problem. Extending these investigations would be an interesting next step. This methodology can also be applied to other combinatorial problems and is not restricted to the dynamic and stochastic knapsack problem, where a similar second objective can be formulated to deal with the chance constraint.

Next, we investigated a combinatorial problem with a more real-world perspective in Chapter 5: an engineering design problem known as the truss optimisation problem was considered. In truss optimisation, the main aim is to minimise the weight of the structure, which is the summation of the weight of truss members. The weight of each member relies on their cross-section area and length. We used bi-level optimisation to consider interactions among different aspects of the optimisation problem. We showed that the upper level could be viewed as a combinatorial problem in deciding whether to include or exclude a member. For small-scale problems, we rigorously investigated truss design using exact methods. Additionally, we developed bio-inspired approaches based on Novelty Search for more complex problems and obtained high-quality solutions. Studying this approach for other structural problems such as plates, shells, and frames could be interesting. Another suggestion could be to reduce the computational cost of the problem.

Then, we introduced the stockpile recovery problem as a real-world scheduling

problem in Chapter 6. Stockpile recovery is a complex problem in the mining industry, where poor schedules can lead to financial penalties for providers. We consider solution construction heuristics to develop a solution step by step to solve the problem subject to technical constraints and using a single reclaimer for the operation. We prioritised minimising penalty fees due to contaminants in delivery over operating costs in comparing two solutions. At first, we considered relaxing some of the technical requirements to make the problem more tractable. We developed deterministic and randomised greedy algorithms and ant colony optimisation to solve the problem. Our findings showed that ACO outperforms other algorithms, and the variant integrated with swap and insert local search operators finds the best solutions.

In Chapter 7, we added more realistic settings to the stockpile recovery problem considering multiple reclaimers and more technical requirements, such as handling different minerals with different directions of reclaiming. We devised a heuristic for solution construction to consider the interaction among reclaimers. We examined deterministic and randomised greedy algorithms and an ant colony optimisation variation. Our algorithms were fine-tuned using automatic parameter tuning method, and we determined the optimal configuration for each instance. Additionally, we developed a local search operator for ant colony optimisation to refine the coarse search. As a result, we concluded that ant colony optimisation is highly promising, outperforming other methods in most instances similar to what we found in the initial study. We can add a human to the loop to revise the algorithms' solutions to facilitate the practical application of the work. Adding dynamic settings to the problem might be interesting to simulate stacking and reclaiming a stockpile, where stacking and reclaiming occur simultaneously in practice. It could be interesting to investigate how to add a threshold for considering trade-offs among objectives to achieve better schedules.

Bibliography

[AAD20] Ali Ahrari, Ali-Asghar Atai, and Kalyanmoy Deb. "A customized bilevel optimization approach for solving large-scale truss design problems". In: *Engineering Optimization* 52.12 (2020), pp. 2062–2079.

[AD16] Ali Ahrari and Kalyanmoy Deb. "An improved fully stressed design evolution strategy for layout optimization of truss structures". In: *Computers & Structures* 164 (2016), pp. 127–144.

[Ang+16] Enrico Angelelli, Thomas Kalinowski, Reena Kapoor, and Martin W P Savelsbergh. "A reclaimer scheduling problem arising in coal stockyard management". In: *Journal of Scheduling* 19.5 (2016), pp. 563–582.

[Ass+20] Hirad Assimi, Oscar Harper, Yue Xie, Aneta Neumann, and Frank Neumann. "Evolutionary Bi-objective Optimization for the Dynamic Chance-Constrained Knapsack Problem Based on Tail Bound Objectives". In: *Proceedings of the European Conference on Artificial Intelligence (ECAI'20)*. Vol. 325. IOS Press, 2020, pp. 307–314.

[Ass+21] Hirad Assimi, Ben Koch, Chris Garcia, Markus Wagner, and Frank Neumann. "Modelling and Optimization of Run-of-Mine Stockpile Recovery". In: *36th Annual ACM Symposium on Applied Computing*. ACM, 2021, pp. 450–458.

[Ass+22a] Hirad Assimi, Ben Koch, Chris Garcia, Markus Wagner, and Frank Neumann. "Run-of-Mine Stockyard Recovery Scheduling and Optimisation for Multiple Reclaimers". In: *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*. SAC '22. Association for Computing Machinery, 2022, pp. 1074–1083. ISBN: 9781450387132. DOI: 10.1145/3477314.3507130.

[Ass+22b] Hirad Assimi, Frank Neumann, Markus Wagner, and Xiaodong Li. "Novelty-Driven Binary Particle Swarm Optimisation for Truss Optimisation Problems". In: *Evolutionary Computation in Combinatorial Optimization*. Ed. by Leslie Pérez Cáceres and Sébastien Verel. Cham: Springer International Publishing, 2022, pp. 111–126. ISBN: 978-3-031-04148-8.

[BGN09] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust Optimization*. Princeton Series in Applied Mathematics. Princeton University Press, 2009, pp. 29–30. ISBN: 9781400831050.

[Bir+10] Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle. "F-Race and iterated F-Race: An overview". In: *Experimental methods for the analysis of optimization algorithms* (2010), pp. 311–336.

[BKM17] Timothy R Brooks, Gaetan K Kenway, and Joaquim R R A Martins. "Undeflected common research model (uCRM): an aerostructural model for the study of high aspect ratio transport aircraft wings". In: *35th AIAA Applied Aerodynamics Conference*. 2017, p. 4456.

[BS07] Hans-Georg Beyer and Bernhard Sendhoff. "Robust optimization–a comprehensive survey". In: *Computer methods in applied mechanics and engineering* 196.33-34 (2007), pp. 3190–3218.

[BT19] Daniel Blado and Alejandro Toriello. "Relaxation analysis for the dynamic knapsack problem with stochastic item sizes". In: *SIAM Journal on Optimization* 29.1 (2019), pp. 1–30.

[CB02] George Casella and Roger L Berger. *Statistical inference*. Vol. 2. Duxbury Press, 2002.

[CC59] Abraham Charnes and William W Cooper. "Chance-constrained programming". In: *Management science* 6.1 (1959), pp. 73–79.

[CF14] Gregory W Corder and Dale I Foreman. *Nonparametric statistics: A step-by-step approach*. John Wiley & Sons, 2014.

[Che16] M Cheng. "A Hybrid Harmony Search algorithm for discrete sizing optimization of truss structure". In: *Automation in Construction* 69 (2016), pp. 21–33.

[CW20] Jonatas B C Chagas and Markus Wagner. "Ants can orienteer a thief in their robbery". In: *Operations Research Letters* 48.6 (2020), pp. 708–714. ISSN: 0167-6377.

[CWM12] Raymond Chiong, Thomas Weise, and Zbigniew Michalewicz. *Variants of evolutionary algorithms for real-world applications*. Springer, 2012.

[Deb+02] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T Meyarivan. "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE Transactions on Evolutionary Computation* 6.2 (Apr. 2002), pp. 182–197. ISSN: 1089-778X. DOI: 10.1109/4235.996017.

[Deb01] Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*. Vol. 16. John Wiley & Sons, 2001.

[DFJ54] George Dantzig, Ray Fulkerson, and Selmer Johnson. "Solution of a large-scale traveling-salesman problem". In: *Journal of the operations research society of America* 2.4 (1954), pp. 393–410.

[DG01] Kalyanmoy Deb and Surendra Gulati. "Design of truss-structures for minimum weight using genetic algorithms". In: *Finite Elements in Analysis and Design* 37.5 (2001), pp. 447–465.

[DLU18] Sadk Ozgur Degertekin, Luciano Lamberti, and I B Ugur. "Sizing, layout and topology design optimization of truss structures using the Jaya algorithm". In: *Applied Soft Computing* 70 (2018), pp. 903–928.

[DLU19] S O Degertekin, L Lamberti, and I B Ugur. "Discrete sizing/layout/topology optimization of truss structures with an advanced Jaya algorithm". In: *Applied Soft Computing* 79 (2019), pp. 363–390.

[Doe+20] Benjamin Doerr, Carola Doerr, Aneta Neumann, Frank Neumann, and Andrew M Sutton. "Optimization of Chance-Constrained Submodular Functions". In: *Proc. of AAAI*. 2020.

[DS04] Marco Dorigo and Thomas Stützle. *Ant colony optimization*. English. Cambridge, MA: MIT Press, 2004. ISBN: 0-262-04219-3.

[DS16] Annelies De Corte and Kenneth Sörensen. "An iterated local search algorithm for water distribution network design optimization". In: *Networks* 67.3 (2016), pp. 187–198.

[ES15] A E Eiben and J E Smith. "Fitness, Selection, and Population Management". In: *Introduction to Evolutionary Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 79–98. ISBN: 978-3-662-44874-8.

[Fen+14] Michael Fenton, Ciaran McNally, Jonathan Byrne, Erik Hemberg, James McDermott, and Michael O'Neill. "Automatic innovative truss design using grammatical evolution". In: *Automation in Construction* 39.C (2014), pp. 59–69.

[FGS16] Marcello Farina, Luca Giulioni, and Riccardo Scattolini. "Stochastic linear model predictive control with chance constraints–a review". In: *Journal of Process Control* 44 (2016), pp. 53–67.

[Fin+13] Vitor C Finotto, Wilson R L da Silva, M Valášek, and P Štemberk. "Hybrid fuzzy-genetic system for optimising cabled-truss structures". In: *Advances in Engineering Software* 62 (2013), pp. 85–96.

[Fis+19] Iztok Fister, Andres Iglesias, Akemi Galvez, Javier Del Ser, Eneko Osaba, Iztok Fister, Matjaž Perc, and Mitja Slavinec. "Novelty search for global optimization". In: *Applied Mathematics and Computation* 347 (2019), pp. 865–881. ISSN: 0096-3003. DOI: https://doi.org/10.1016/j.amc.2018.11.052. URL: https://www.sciencedirect.com/science/article/pii/S0096300318310269.

[FLP16] Galvao Diana F., Joel Lehman, and Urbano Paulo. "Novelty-Driven Particle Swarm Optimization". In: *Artificial Evolution*. Ed. by Bonnevay Stéphane, Pierrick Legrand, Monmarché Nicolas, Lutton Evelyne, and Schoenauer Marc. Springer, 2016, pp. 177–190.

[Gao+18] Wanru Gao, Tobias Friedrich, Frank Neumann, and Christian Hercher. "Randomized greedy algorithms for covering problems". In: *Genetic and Evolutionary Computation Conference*. 2018, pp. 309–315.

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

[GR10] Vineet Goyal and R Ravi. "A PTAS for the chance-constrained knapsack problem with random item sizes". In: *Operations Research Letters* 38.3 (2010), pp. 161–164.

[Has07] OĞUZHAN Hasancebi. "Optimization of truss bridges within a specified design domain using evolution strategies". In: *Engineering Optimization* 39.6 (2007), pp. 737–756.

[HE01] OĞUZHAN Hasancebi and F Erbatur. "Layout optimization of trusses using improved GA methodologies". In: *Acta mechanica* 146.1 (2001), pp. 87–107.

[HE02] Oğuzhan Hasançebi and Fuat Erbatur. "Layout optimisation of trusses using simulated annealing". In: *Advances in Engineering Software* 33.7 (2002), pp. 681–696.

[Ho-+16] V Ho-Huu, T Nguyen-Thoi, T Vo-Duy, and T Nguyen-Trang. "An adaptive elitist differential evolution for optimization of truss structures with discrete design variables". In: *Computers & Structures* 165.C (2016), pp. 59–75.

[HPW04] S He, E Prempain, and Q H Wu. "An improved particle swarm optimizer for mechanical design optimization problems". In: *Engineering optimization* 36.5 (2004), pp. 585–605.

[HS04] Holger H Hoos and Thomas Stützle. *Stochastic local search: Foundations and applications*. Elsevier, 2004.

[HS05] Holger H. Hoos and Thomas Stützle. "2 - SLS METHODS". In: *Stochastic Local Search*. Ed. by Holger H. Hoos and Thomas Stützle. The Morgan Kaufmann Series in Artificial Intelligence. San Francisco: Morgan Kaufmann, 2005, pp. 61–112. ISBN: 978-1-55860-872-6.

[Ibr+14] Maksud Ibrahimov, Arvind Mohais, Sven Schellenberg, and Zbigniew Michalewicz. "Scheduling in iron ore open-pit mining". In: *The International Journal of Advanced Manufacturing Technology* 72.5-8 (2014), pp. 1021–1037.

[ILM17] Md. Jakirul Islam, Xiaodong Li, and Yi Mei. "A time-varying transfer function for balancing the exploration and exploitation ability of a binary PSO". In: *Applied Soft Computing* 59 (2017), pp. 182–196.

[JHE13] K Jupp, T J Howard, and J E Everett. "Role of pre-crusher stockpiling for grade control in iron ore mining". In: *Applied Earth Science* 122.4 (2013), pp. 242–255.

[Jin+16] Han Jinil, Kyungsik Lee, Lee Chungmok, Choi Ki-Seok, and Park Sungsoo. "Robust optimization approach for a chance-constrained binary knapsack problem". In: *Mathematical Programming* 157.1 (2016), pp. 277–296. ISSN: 1436-4646.

[Jin19] Ce Jin. "An improved FPTAS for 0-1 knapsack". In: *arXiv preprint arXiv:1904.09562* (2019).

[KAD05] Rafal Kicinger, Tomasz Arciszewski, and Kenneth De Jong. "Evolutionary computation and structural design: A survey of the state-of-the-art". In: *Computers & structures* 83.23-24 (2005), pp. 1943–1978.

[KE95] James Kennedy and Russell Eberhart. "Particle swarm optimization". In: *International Conference on Neural Networks (ICNN)*. Vol. 4. 1995, pp. 1942–1948.

[Kha+20] Hamid Khayyam, Ali Jamali, Hirad Assimi, and Reza N Jazar. "Genetic programming approaches in design and optimization of mechanical engineering applications". In: *Nonlinear approaches in engineering applications*. Springer, 2020, pp. 367–402.

[KN08] Olivier Klopfenstein and Dritan Nace. "A robust approach to the chance-constrained knapsack problem". In: *Oper. Res. Lett.* 36.5 (2008), pp. 628–632. DOI: 10.1016/j.orl.2008.03.006.

[Kos14] Stefanie Kosuch. "Approximability of the two-stage stochastic knapsack problem with discretely distributed weights". In: *Discrete Applied Mathematics* 165 (2014), pp. 192–204.

[KPP04] Hans Kellerer, Ulrich Pferschy, and David Pisinger. "Introduction to NP-Completeness of knapsack problems". In: *Knapsack problems*. Springer, 2004, pp. 483–493.

[KRT00] Jon Kleinberg, Yuval Rabani, and Éva Tardos. "Allocating bandwidth for bursty connections". In: *SIAM Journal on Computing* 30.1 (2000), pp. 191–217.

[KT09] A Kaveh and S Talatahari. "A particle swarm ant colony optimization for truss structures with discrete variables". In: *Journal of Constructional Steel Research* 65.8-9 (2009), pp. 1558–1568.

[Lee+05] Kang Seok Lee, Zong Woo Geem, Sang-ho Lee, and Kyu-woong Bae. "The harmony search heuristic algorithm for discrete structural optimization". In: *Engineering Optimization* 37.7 (2005), pp. 663–684.

[LHL09] L J Li, Z B Huang, and F Liu. "A heuristic particle swarm optimization method for truss structures with discrete variables". In: *Computers & Structures* 87.7-8 (2009), pp. 435–443.

[Li+16a] Guijie Li, Zhenzhou Lu, Luyi Li, and Bo Ren. "Aleatory and epistemic uncertainties analysis based on non-probabilistic reliability and its kriging solution". In: *Applied Mathematical Modelling* 40.9-10 (2016), pp. 5703–5716.

[Li+16b] Xiaodong Li, Michael G Epitropakis, Kalyanmoy Deb, and Andries Engelbrecht. "Seeking multiple solutions: an updated survey on niching methods and their applications". In: *IEEE Transactions on Evolutionary Computation* 21.4 (2016), pp. 518–538.

[Li+19] Siyi Li, Marco de Werk, Louis St-Pierre, and Mustafa Kumral. "Dimensioning a stockpile operation using principal component analysis". In: *International Journal of Minerals, Metallurgy and Materials* 26.12 (2019), pp. 1485–1494.

[Liu+13] Bo Liu, Qingfu Zhang, Francisco V Fernandez, and Georges Gielen. "An efficient evolutionary algorithm for chance-constrained bi-objective stochastic optimization". In: *IEEE Transactions on Evolutionary Computation* 17.6 (2013), pp. 786–796.

[LL15] Zhuangzhi Li and Zukui Li. "Chance constrained planning and scheduling under uncertainty using robust optimization approximation". In: *IFAC-PapersOnLine* 48.8 (2015), pp. 1156–1161.

[LM10] Tien-Fu Lu and Maung Thi Rein Myo. "Optimization of reclaiming voxels for quality grade target with reclaimer minimum movement". In: *11th International Conference on Control Automation Robotics & Vision*. 2010, pp. 341–345.

[LM11] Tien-Fu Lu and Maung Thi Rein Myo. "Optimal stockpile voxel identification based on reclaimer minimum movement for target grade". In: *International Journal of Mineral Processing* 98.1-2 (2011), pp. 74–81.

[LMS03] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. "Iterated local search". In: *Handbook of metaheuristics*. Springer, 2003, pp. 320–353.

[Lob+19] Fran Sérgio Lobato, Márcio Aurelio da Silva, Aldemir Aparecido Cavalini, and Valder Steffen. "Reliability-Based Robust Optimization Applied to Engineering System Design". In: *Computational Intelligence, Optimization and Inverse Problems with Applications in Engineering*. Springer, 2019, pp. 29–52.

[Lóp+16] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. "The irace package: Iterated racing for automatic algorithm configuration". In: *Operations Research Perspectives* 3 (2016), pp. 43–58.

[LS08] Joel Lehman and Kenneth O Stanley. "Exploiting open-endedness to solve problems through the search for novelty." In: *ALIFE*. 2008, pp. 329–336.

[LS11a] Joel Lehman and Kenneth O Stanley. "Abandoning objectives: Evolution through the search for novelty alone". In: *Evolutionary computation* 19.2 (2011), pp. 189–223.

[LS11b] Joel Lehman and Kenneth O. Stanley. "Improving evolvability through novelty search and self-adaptation". In: *2011 IEEE Congress of Evolutionary Computation (CEC)*. 2011, pp. 2693–2700. DOI: 10.1109/CEC.2011.5949955.

[LYT15] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. "Constrained Novelty Search: A Study on Game Content Generation". In: *Evol. Comput.* 23.1 (Mar. 2015), pp. 101–129. ISSN: 1063-6560. DOI: 10.1162/EVCO_a_00123. URL: https://doi.org/10.1162/EVCO_a_00123.

[MA94] Zbigniew Michalewicz and Jarosław Arabas. "Genetic algorithms for the 0/1 knapsack problem". In: *International Symposium on Methodologies for Intelligent Systems*. 1994, pp. 134–143.

[Mar+19] Aritz D. Martinez, Eneko Osaba, Izaskun Oregi, Iztok Fister, Iztok Fister, and Javier Del Ser. "Hybridizing Differential Evolution and Novelty Search for Multimodal Optimization Problems". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '19. Prague, Czech Republic: Association for Computing Machinery, 2019, pp. 1980–1989. ISBN: 9781450367486. DOI: 10.1145/3319619.3326799. URL: https://doi.org/10.1145/3319619.3326799.

[McD98] Colin McDiarmid. "Concentration". In: *Probabilistic Methods for Algorithmic Discrete Mathematics*. Ed. by Habib Michel, Colin McDiarmid, Ramirez-Alfonsin Jorge, and Reed Bruce. Springer Berlin Heidelberg, 1998, pp. 195–248. ISBN: 978-3-662-12788-9. DOI: 10.1007/978-3-662-12788-9{_}6.

[Mit98] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

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

[NGS05] Efstratios Nikolaidis, Dan M Ghiocel, and Suren Singhal. *Types of uncertainty in design decision making*. CRC Press, New York, 2005.

[NN20] Aneta Neumann and Frank Neumann. "Optimising monotone chance-constrained submodular functions using evolutionary multi-objective algorithms". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2020, pp. 404–417.

[NS19] Frank Neumann and Andrew M Sutton. "Runtime analysis of the (1+1) evolutionary algorithm for the chance-constrained knapsack problem". In: *Proceedings of the 15th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*. 2019, pp. 147–153.

[NSW09] Frank Neumann, Dirk Sudholt, and Carsten Witt. "Analysis of different MMAS ACO algorithms on unimodal functions and plateaus". In: *Swarm Intelligence* 3.1 (2009), pp. 35–68.

[NW06] Frank Neumann and Ingo Wegener. "Minimum spanning trees made easier via multi-objective optimization". In: *Natural Computing* 5.3 (2006), pp. 305–319.

[NW10] Frank Neumann and Carsten Witt. "Bioinspired Computation in Combinatorial Optimization". In: Natural Computing Series (2010). DOI: 10.1007/978-3-642-16544-3.

[NW21] Frank Neumann and Carsten Witt. "Runtime Analysis of Single-and Multi-Objective Evolutionary Algorithms for Chance Constrained Optimization Problems with Normally Distributed Random Variables". In: *arXiv preprint arXiv:2109.05799* (2021).

[NYB12] Trung Thanh Nguyen, Shengxiang Yang, and Juergen Branke. "Evolutionary dynamic optimization: A survey of the state of the art". In: *Swarm and Evolutionary Computation* 6 (2012), pp. 1–24.

[PAS09] Bernardo K Pagnoncelli, Shapiro Ahmed, and Alexander Shapiro. "Sample average approximation method for chance constrained programming: theory and applications". In: *Journal of optimization theory and applications* 142.2 (2009), pp. 399–416.

[PB18] Natee Panagant and Sujin Bureerat. "Truss topology, shape and sizing optimization by fully stressed design based on hybrid grey wolf optimization and adaptive differential evolution". In: *Engineering Optimization* 50.10 (2018), pp. 1645–1661.

[Pin16] Michael L Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 2016.

[Pol+14] Sergey Polyakovskiy, Mohammad Reza Bonyadi, Markus Wagner, Zbigniew Michalewicz, and Frank Neumann. "A comprehensive benchmark set and heuristics for the traveling thief problem". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2014*. 2014, pp. 477–484.

[Qia+17] Chao Qian, Jing-Cheng Shi, Yang Yu, and Ke Tang. "On Subset Selection with General Cost Constraints". In: *International Joint Conference on Artificial Intelligence, IJCAI 2017*. 2017, pp. 2613–2619.

[QYZ15] Chao Qian, Yang Yu, and Zhi-Hua Zhou. "Subset Selection by Pareto Optimization". In: *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems, NIPS 2015*. 2015, pp. 1774–1782.

[Rao95] G Visweswara Rao. "Optimum designs for transmission line towers". In: *Computers & structures* 57.1 (1995), pp. 81–92.

[RK92] S Rajeev and C S Krishnamoorthy. "Discrete optimization of structures using genetic algorithms". In: *Journal of Structural Engineering* 118.5 (1992), pp. 1233–1250.

[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 2018*. Lecture Notes in Computer Science. 2018, pp. 158–169.

[Roo+19] Vahid Roostapour, Aneta Neumann, Frank Neumann, and Tobias Friedrich. "Pareto optimization for subset selection with dynamic cost constraints". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 2354–2361.

[Sam+17] Mehran Samavati, Daryl Essam, Micah Nehring, and Ruhul Sarker. "A local branching heuristic for the open pit mine production scheduling problem". In: *European Journal of Operational Research* 257.1 (2017), pp. 261–271.

[Seb+11] Guclu Seber, Hongjun Ran, Taeqoo Nam, Joseph Schetz, and Dimitri Mavris. "Multidisciplinary design optimization of a truss braced wing aircraft with upgraded aerodynamic analyses". In: *29th AIAA Applied Aerodynamics Conference*. 2011, p. 3179.

[SH00] Thomas Stützle and Holger H Hoos. "MAX–MIN Ant System". In: *Future Generation Computer Systems* 16.8 (2000), pp. 889–914. ISSN: 0167-739X.

[Sip92] Michael Sipser. "The history and status of the P versus NP question". In: *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*. 1992, pp. 603–618.

[Sip97] M Sipser. "Introduction to the theory of computation, PWS Pub". In: *Co., Boston* (1997).

[SP97] Rainer Storn and Kenneth Price. "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces". In: *Journal of global optimization* 11.4 (1997), pp. 341–359.

[SS15] Masoud Soleymani Shishvan and Javad Sattarvand. "Long term production planning of open pit mines by ant colony optimization". In: *European Journal of Operational Research* 240.3 (2015), pp. 825–836.

[Top83] B H V Topping. "Shape optimization of skeletal structures: a review". In: *Journal of Structural Engineering* 109.8 (1983), pp. 1933–1951.

[UO19] Ozgur Unsal and Ceyda Oguz. "An exact algorithm for integrated planning of operations in dry bulk terminals". In: *Transportation Research Part E: Logistics and Transportation Review* 126 (2019), pp. 103–121.

[WC95] S J Wu and P T Chow. "Steady-state genetic algorithms for discrete optimization of trusses". In: *Computers & Structures* 56.6 (1995), pp. 979–991.

[Xie+19] Yue Xie, Oscar Harper, Hirad Assimi, Aneta Neumann, and Frank Neumann. "Evolutionary algorithms for the chance-constrained knapsack problem". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019*. 2019, pp. 338–346.

[Xie+21] Yue Xie, Aneta Neumann, Frank Neumann, and Andrew M Sutton. "Runtime analysis of RLS and the (1+ 1) EA for the chance-constrained knapsack problem with correlated uniform weights". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2021, pp. 1187–1194.

[XNN20] Yue Xie, Aneta Neumann, and Frank Neumann. "Specific single-and multi-objective evolutionary algorithms for the chance-constrained knapsack problem". In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 2020, pp. 271–279.

[XNN21a] Yue Xie, Aneta Neumann, and Frank Neumann. "Heuristic Strategies for Solving Complex Interacting Large-Scale Stockpile Blending Problems". In: *2021 IEEE Congress on Evolutionary Computation (CEC)*. 2021, pp. 1288–1295.

[XNN21b] Yue Xie, Aneta Neumann, and Frank Neumann. "Heuristic Strategies for Solving Complex Interacting Stockpile Blending Problem with Chance Constraints". In: *Genetic and Evolutionary Computation Conference*. ACM, 2021, pp. 1079–1087. ISBN: 9781450383509.

[Zha+05] Yan-Nian Zhang, Ping Liu, Bin Liu, Chao-Yan Zhu, and Yi Li. "Application of improved hybrid genetic algorithm to optimized design of architecture structures". In: *Huanan Ligong Daxue Xuebai(Ziran Kexue Ban)/ Journal of South China University of Technology(Natural Science Edition)(China)* 33.3 (2005), pp. 69–72.