DOCTORAL THESIS

Exact and Heuristic Approaches for Multi-component Optimisation Problems

Author:
Junhua WU

Principal Supervisor:
Prof. Frank NEUMANN

Co-Supervisor:
Dr Sergey POLYAKOVSKIY

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

in the

Optimisation and Logistics
School of Computer Science

THE UNIVERSITY of ADELAIDE

July 4, 2018

Declaration of Authorship

I, Junhua WU, certify that this thesis titled, "Exact and Heuristic Approaches for Multi-component Optimisation Problems", 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 Scheme (RTS) Scholarship.

Signed: _____

Date: _____

"The universe is just this swirling mass of atoms, forming clumps of various kinds of things, and dissolving. Most of those atoms don't get to be alive at all. Most of those atoms don't get to be a person, falling in love, seeing sunsets, eating ice cream. It's extraordinarily lucky of us to be in this select, fortunate few."

Prof. Shelly Kagan

THE UNIVERSITY OF ADELAIDE

Abstract

Faculty of Engineering, Computer and Mathematical Sciences (ECMS)
School of Computer Science

Doctor of Philosophy

Exact and Heuristic Approaches for Multi-component Optimisation Problems

by Junhua WU

Modern real world applications are commonly complex, consisting of multiple subsystems that may interact with or depend on each other. Our case-study about wave energy converters (WEC) for the renewable energy industry shows that in such a multi-component system, optimising each individual component cannot yield global optimality for the entire system, owing to the influence of their interactions or the dependence on one another. Moreover, modelling a multi-component problem is rarely easy due to the complexity of the issues, which leads to a desire for existent models on which to base, and against which to test, calculations. Recently, the travelling thief problem (TTP) has attracted significant attention in the Evolutionary Computation community. It is intended to offer a better model for multi-component systems, where researchers can push forward their understanding of the optimisation of such systems, especially for understanding of the interconnections between the components. The TTP interconnects with two classic $\mathcal{NP}$-hard problems, namely the travelling salesman problem and the 0-1 knapsack problem, via the transportation cost that non-linearly depends on the accumulated weight of items. This non-linear setting introduces additional complexity. We study this nonlinearity through a simplified version of the TTP - the packing while travelling (PWT) problem, which aims to maximise the total reward for a given travelling tour. Our theoretical and experimental investigations demonstrate that the difficulty of a given problem instance is significantly influenced by adjusting a single parameter, the renting rate, which prompted our method of creating relatively hard instances using simple evolutionary algorithms. Our further investigations into the PWT problem yield a dynamic programming (DP) approach that can solve the problem in pseudo polynomial time and a corresponding approximation scheme. The experimental investigations show that the new approaches outperform the state-of-the-art ones. We furthermore propose three exact algorithms for the TTP, based on the DP of the PWT problem. By employing the exact DP for the underlying PWT problem as a subroutine, we create a novel indicator-based hybrid evolutionary approach for a new bi-criteria formulation of the TTP. This hybrid design takes advantage of the DP approach, along with a number of novel indicators and selection mechanisms to achieve better solutions. The results of computational experiments show that the approach is capable to outperform the state-of-the-art results.

Acknowledgements

It has been a great honour to work with the Optimisation and Logistics team at the School of Computer Science, the University of Adelaide. As a worldwide leader position in this domain, this team provided me with the opportunities to learn not only techniques such as evolutionary computation, hybrid approaches, mining big data, machine learning etc., but more importantly the methodologies of research and philosophy of being a researcher. I believe the latter has become a part of my personality which will continue to benefit me beyond academia.

I therefore would like to express my heartfelt gratitude to the leader of the team and my principal supervisor, Professor Frank Neumann. I am most grateful to him for inviting me into his team and supporting the scholarships that enabled me to focus on my studies. He also devoted hours his time to guiding and advising me on the path to achieving sound research. Without his warm support, I would not have had the chance to undertake such an inspiring journey.

I would also thank my two key advisors and co-authors, Dr Sergey Polyakovskiy and Dr Markus Wagner, who both generously shared their wisdom, advices and feedback, which have been crucial for forming my understanding of research. They provided selfless support whenever I needed aid and gave me great insights into the research topics. I have very appreciated of working with both of them.

The best experiences in my academic life have been the collaborations and sharing of ideas, which have constituted the main sources of inspiration for my studies. I would like to express my genuine thanks to the following people who either advised me on various topics, acted as my co-authors, or kindly shared their studies with our team: Dr Bradley Alexander, Dr Yaneli Ameca Alducin, Mahmoud Bokhari, Dr Mohammadreza Bonyadi, Professor Benjamin S. Cazzolato, Christopher Denison, Dr Boyin Ding, Dr Wanru Gao, Dr Mingyu Guo, Maryam Hasani, Dr Zhibin Liao, Dr Lingqiao Liu, Dr Xiang Liu, Dr Zhi Lu, Professor Zbigniew Michalewicz, Dr Samadhi Nallaperuma, Aneta Neumann, Dr Mojgan Pourhassan, Dr Shayan Poursoltan, Vahid Roostapour, Nataliia Y. Sergiienko, Slava Shekh, Professor Chunhua Shen, Feng Shi, Dr Qinfeng Shi, Professor Martin Skutella, Professor Leen Stougie, Raymond Tran, Dr Peng Wang and all the countless others who have worked to make this team so special.

Moreover, I would like to thank the Australia Research Council, the Commonwealth Government, the School of Computer Science of the University of Adelaide, and the Genetic and Evolutionary Computation Conference (GECCO) for funding the scholarships that have covered my tuition fees, living expenses and the conference travels; the researchers and practitioners who have contributed to the foundation of this domain in which I can study; the anonymous reviewers of my papers and the examiners of this thesis for their precious feedback; Miss Alison-Jane Hunter for her diligent language editing and proofreading of this thesis; and the facilitators in our school and the Adelaide Graduate Centre for their assistance with ensuring the progress of my PhD schedule.

Lastly but not least, I offer my wholehearted gratitude to my wife Jasmine, my parents Yinguan and Shengguan, my parents-in-law Cuie and Yuan, and my dear relatives and friends. Without your understanding and support, I would not have been able to go this far. Thank you all for having faith in me.

Contents

List of Figures

List of Tables

List of Abbreviations

BCP	Bound Constrained Optimisation Problem
BFS	Brute Force Search
BIB	Branch-Infer and Bound
BnB	Branch and Bound
CMA-ES	Co-Variance Matrix Adaptation Based Evolutionary Strategy
CP	Constraint Programming
CPU	Central Processing Unit
CSPs	Constraint Satisfaction Problems
DP	Dynamic Programming
EA	Evolutionary Algorithm
ERX	Edge Recombination Crossover
FPTAS	Fully Polynomial Approximation Scheme
GB	Gigabyte
GECCO	The Genetic and Evolutionary Computation Conference
HPC	High-Performance Computing
JSP	Job Shop Scheduling Problem
KP	Knapsack Problem
KPI	Key Performance Indicator
LP	Linear Programming Problems
MIP	Mixed Integer Programming
MMAS	Max-Min Ant System
MPX	Maximum Preservative Crossover
MST	Minimum Spanning Tree
NE	Nonlinear Equations Problem
NLP	Nonlinear Programming Problems
NLSP	Nonlinear Least-Squares Problem
$\mathcal{NP}$	Nondeterministic Polynomial Time
PWT	Packing While Travelling
RAM	Random-Access Memory
RCW	Relative Capture Width
RT	Running Time
SEAL	International Conference On Simulated Evolution and Learning
TSP	Travelling Salesman Problem
TTP	Travelling Thief Problem
VRP	Vehicle Routing Problem
WEC	Wave Energy Converter

List of Symbols

C	capacity[1]
D	distance matrix
d_{ij}	the distance between city i and city j
M	a set of items
M_i	the set of items at city i
m	the size of M or the m-th item
N	a set of cities
n	the size of N or the n-th city
p_i	the profit of item i
P_L	the lower bound of profits[1]
P_U	the upper bound of profits[1]
R	renting rate[1]
v_{x_i}	the velocity when the thief/vehicle is leaving from city x_i
v_{max}	maximal velocity[1]
v_{min}	minimal velocity[1]
w_i	the weight of item i
W_L	the lower bound of weights[1]
W_U	the upper bound of weights[1]
W_{x_i}	the weight carried by the thief/vehicle when leaving from city x_i
ν	speed coefficient
π	a tour or route
ρ	a packing plan

[1] Constant

TO MY BELOVED FAMILY.

Chapter 1

Introduction

Nowadays, real-world applications are becoming ever more complex. For instance, in the paradigm of modern business management, individual commercial activities no longer compete as solely autonomous entities (namely *silos*), but rather as supply chains [92]. Not only internal functional departments (such as Finance, Production, Sales, Logistics and etc.), but also external suppliers and/or distributors are tightly integrated in order to pursue the success of the business as a whole. Figure 1.1 depicts a framework of a supply chain management structure that is guided by such a philosophy, in which materials, products, services, information and so on are controlled and managed across both multi-functional silos within the company and the various corporate silos outside. The purpose of designing such a complex multi-component system is that the overall performance of the system is presumed to be better than the loose aggregate of the silos, as synergy of the integration is expected to increase productivity.

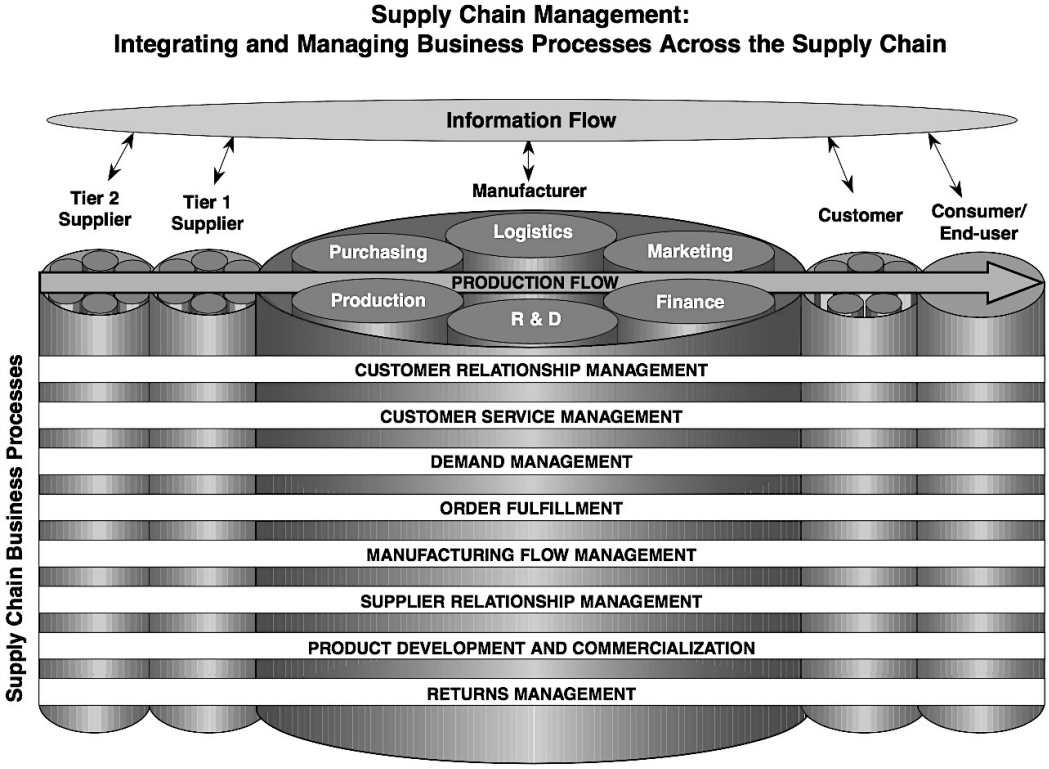


FIGURE 1.1: Supply Chain Management Structure [39]

Similar circumstances occur in industrial engineering. As an example, we will present our case-study, wave energy converter (WEC) optimisation, in Chapter 4. In this study, we try to optimise the overall energy absorption of an array of WECs, which are the devices that capture and convert wave energy to electricity or that power a reverse osmosis desalination plant to create potable water [106]. In order to achieve this goal, not only the parameters of each individual WEC (such as the shape and/or dimensions, stiffness, damping coefficient, and etc.) are to be optimised according to the hydrological frequencies in the ocean area to be deployed, but also the hydrodynamic interactions among WECs have to be taken into consideration, because the interactions have a significant impact on the overall performance of the array, i.e. it can be constructed or deconstructed by the interactions [153]. Therefore, only optimising each WEC (silo) cannot yield the overall optimality of an array.

Both of the above systems can be regarded as the *multi-component systems* [9], each of which is composed of multiple components that may interact with and/or interdependent on each other. More importantly, such a system has to be regarded as an entirety when it is measured, analysed, optimised and so on.

In general, when we conduct activities (e.g. measurement, analysis, optimisation, and so forth) with a system, we follow a universal problem solving approach of two steps: step 1) create a model of the problem or system; step 2) study the model and generate solutions [120], namely:

$$Problem \longrightarrow Model \longrightarrow Solutions. \tag{1.1}$$

However, for a multi-component system, it is unlikely to be a trivial matter to create an accurate model, due to the complexities within the system itself. In other words, modelling the multi-component problems may either be impossible or extremely expensive. This leads to a need for existent models or what are more commonly called the *benchmark problems*.

A benchmark problem is a well-defined model, which can be treated as a unified platform with which researchers can experiment and compare different algorithms fairly. This platform not only saves the effort of modelling from scratch, but also helps to improve verifiability - a ubiquitous requirement of scientific publications [120]. Verifiability requires that researchers must demonstrate the completeness of the entire process of problem-solving in their publications, meaning the processes and results are rigorously presented and compared, so that the readers can replicate the approaches taken. By working on an existing model instead of starting from scratch, researchers can compare their work with others' on the same benchmark problem more easily. Famous examples of benchmark problems include the travelling salesman problem (TSP) [34], knapsack problem (KP) [111], minimum spanning tree (MST) [68], constraint satisfaction problems (CSPs) [166], job shop scheduling problem (JSP) [60], linear programming problems (LP) [31, 18], nonlinear programming problems (NLP) [11, 16], bound constrained optimisation problem (BCP) [17], the nonlinear equations problem (NE) [131], nonlinear least-squares problem (NLSP) [10, 83], etc. Researchers usually organise the instances of the problems into various benchmark libraries, so that newcomers can easily start to work with them. Examples include OR-Library [12], TSPLIB [145], CSPLIB [62], PSPLIB [88], SATLIB [79], MVF [1], CUTEst [67], etc.

Despite their academic purpose, most of the current benchmark problems often

fail to model real-world, multi-component systems well, as in general they concentrate on the individual silos instead of measuring the system as a whole. For example, a delivery system typically contains at least two parts: a route/tour planning system and a packing system for organising packages to be delivered, which may be correspondingly represented by the TSP and KP respectively. However, such models do not reflect the interdependence between the tour planning system and the packing system, even though this factor may influence the overall performance of the delivery system significantly. In Chapter 2, we will elaborate a real-world example of such a complex delivery system from a water tank manufacturer in Australia [159], which demonstrates that the interdependences have yet to be taken into account and solving each silo separately in a complex system cannot yield overall optimality.

Meanwhile, some benchmarks indeed take real-world complexities into consideration. For example, the vehicle routing problem (VRP) [95] considers the set of routes for an entire fleet of vehicles to traverse, instead of a single route as in the setting of the TSP. This means that each route of a vehicle has to be considered as a subcomponent to optimise. However, the complexity of this problem is still limited due to its homogeneous subcomponents. Some benchmarks [163, 99, 134] pay special attention to the large size problems formed by interaction between various subcomponents, on which Cooperative Coevolution or other bi-level optimisation approaches [141, 155, 133] are experimented. Nevertheless, the subcomponents mostly are derived from simple test functions.

In 2012, Michalewicz [120] highlighted the gap between the current academical benchmark problems and the real-world multi-component systems, later he and his colleagues proposed a new benchmark problem: the travelling thief problem (TTP) [24], aiming to narrow the gap. Instead of modelling the tour planning part and the packing part separately, the TTP seeks to represent the complex delivery system as a whole. The two disparate $\mathcal{NP}$-hard components are organically bound together in this benchmark by the cost of transportation, so that dealing with the interdependence becomes crucial for solving the problem. In order to tackle a multi-component problem, one might consider the divide and conquer paradigm, which divides the problem into subcomponents and solves them independently. However, the TTP presents a multi-component benchmark problem showing that its two sub-problems (TSP and KP) cannot be easily separated, which makes it uniquely hard to solve.

Thus far, many approximate approaches have been proposed for the TTP [139, 25, 117, 116, 118, 56, 160, 69, 168, 51, 50, 52, 113, 53, 104, 20]. However, to the best of our knowledge, all of them focus on utilising existing heuristic approaches (such as local search [80], simulated annealing [86], tabu search [64], genetic algorithm [123], genetic programming [167], memetic algorithm [127], swarm intelligence [37], etc.), incorporating these with well-studied operators of the TSP and KP (such as 2-opt [38], MPX [96], ERX [96], Bit-flip, etc.) or the slight variations of such operators. Operators or approaches that are specifically designed under the guidance of the theoretical understanding of the TTP remain elusive.

In the journey of pursuing the solutions of a problem, the theoretical understanding of the problem usually plays a vital role. For example, in the research of TSP, Dantzig et al. [40] proposed a method to solve a TSP instance including 49 cities in 1950s, based on the understanding that the TSP problem could be transformed into

an integer linear programming (IP [151]) problem and they used the cutting-plane method [151] to solve this instance. In 1960s, a method of calculating bounds of TSP instances was developed according to the understanding that the length of an optimal tour of a TSP is at most twice the weight of the minimum spanning tree of the TSP graph [3]. Christofides [30] therefore developed an approximate algorithm that guarantees its solutions within a factor of $3/2$ of the optimal solution length. Theoretical understanding helps in the study of heuristic approaches as well. For example, Kötzing et al. [89] recently proved that a local optimum gives a good approximation for the Euclidean TSP, which is based on the understanding that the 2-OPT [38], a famous heuristic operator, shown polynomial bounds on the expected number of steps until 2-OPT reaching the local optimum. They therefore successfully transfer the finding to ACO and improve the performance of the algorithms.

However, the theoretical understanding of the TTP is not easily gained. On the one hand, no exact approach of the TTP so far has been proposed except ours (which will be introduced in Chapter 7) to the best of our knowledge. On the other hand, the TTP is hard not only because of the intertwinement of the two components but also due to the nonlinearity of the transportation cost. The transportation cost is the mathematical factor that connects the two subcomponents in the TTP, while it nonlinearly depends on the weight of the collected items and the overall gain of the packing component depends on the profit from collected items and the transportation cost. This nonlinearity distinguishes the KP component of the TTP from the classical knapsack problems that are characterised by the constraint of weight (namely the capacity of the knapsack), which therefore introduces an additional complexity.

In order to understand more in this nonlinearity of the TTP, Polyakovskiy et al. [140] introduce a simplified version of the TTP: the packing while travelling (PWT) problem, in which the route or tour is a predefined constant. In other words, they only pay attention to the packing part in the TTP. Nevertheless, their research shows that the problem is $\mathcal{NP}$-hard even without the capacity constraint usually imposed on the knapsack (i.e. any combination of items is feasible). On the other hand, previous studies of the heuristics of the TTP indicate that the packing component in general has a greater impact on the overall reward when a near optimal tour is provided [56, 140]. Therefore one of the efficient approaches for solving the TTP is to evolve the packing plan while a set of near optimal tours are provided by a TSP solver.

Putting all the above together, it is reasonable to believe that the PWT problem might be the key to the theoretical understanding of the TTP. In our study, we conduct further investigations on the PWT problem and the TTP, trying to find out: What makes a PWT or TTP instance hard or easy? What essentially distinguishes the PWT problem from classic knapsack problems? Can we find the optimal solutions of the PWT problem efficiently (in pseudo polynomial time) similar to the traditional knapsack problems? Is there any exact approach for the TTP? How can we utilise the exact approaches or the knowledge of such approaches to facilitate and improve the approximate approaches on both its single objective form and the bi-objective form? And so on.

The remainder of this thesis is organised as follows:

Chapters 2 and 3 briefly introduce the backgrounds and prerequisites of the studies discussed and conducted in this thesis. In Chapter 2 we first revisit the general definitions of optimisation problems, and then discuss the relationship between real-world complex systems and the travelling thief problem in greater detail. The formal settings and formulations of the TTP and the PWT problems are also presented in this chapter. In Chapter 3 we review some approaches for optimisation problems. The popular exact and heuristic algorithms such as dynamic programming, branch and bound method, local search, evolutionary algorithms, etc. are briefly introduced. In addition, we also provide a concise literature review of the approaches for the TTP.

Chapter 4 presents our case-study of the optimisation of a real-world complex wave energy system, where the interactions of individual silos have to be considered for the overall optimisation to be successful. Due to the circumstances, the modelling and optimisation of the system are challenging. We demonstrate our novel approximation of this model and the optimisation based upon it.

The studies for the PWT problem are given in Chapters 5 and 6. The study presented in Chapter 5 focuses on an important parameter called the renting rate, which connects the profit from selected items and the nonlinear transportation costs. We then show, using theoretical and experimental investigations, how the renting rate affects the number of items that can be eliminated by the simple pre-processing scheme, hence influencing the overall hardness of a particular PWT/TTP instance. Based on this discovery, we construct instances in a theoretical and experimental way, whereby a simple baseline evolutionary algorithm fails to obtain an optimal solution. Chapter 6 proposes an exact dynamic programming approach of the PWT, which considers the items in the order in which they appear on the route that needs to be travelled and applies dynamic programming, as for the classic knapsack problem. A corresponding relaxation using appropriate rounding of the dynamic programming is also investigated.

Our studies of the travelling thief problem are provided in Chapters 7 and 8. In Chapter 7, we propose three exact approaches for the TTP, formed from an extension of the findings described in the previous chapters (Chapters 5 and 6), which use a dynamic programming approach, the branch and bound search, and the constraint programming. Furthermore, we measured the existing approximate approaches for the TTP based on the results of the best exact approach for some small TTP instances. We then present a multi-objective version of the TTP in Chapter 8, where the goal is to minimise the weight and maximise the overall benefit of a solution. We present a hybrid evolutionary approach for the bi-objective TTP that uses the dynamic programming approach for the underlying PWT problem as a subroutine. The approach is adapted from the basic indicator-based evolutionary algorithm (IBEA [182]) incorporating with a series of novel customised indicators and parent selections in order to encourage the synergy of both the evolutionary algorithm and the dynamic programming approach.

The content of Chapters 4, 5, 6, 7 and 8 are based on the published or submitted papers, as follows respectively:

- J. Wu, S. Shekh, N. Y. Sergiienko, B. S. Cazzolato, B. Ding, F. Neumann, and M. Wagner. "Fast and Effective Optimisation of Arrays of Submerged Wave Energy Converters". In: *Proceedings of the 2016 on Genetic and Evolutionary*

Computation Conference, Denver, CO, USA, July 20 - 24, 2016. 2016, pp. 1045–1052

- J. Wu, S. Polyakovskiy, and F. Neumann. "On the Impact of the Renting Rate for the Unconstrained Nonlinear Knapsack Problem". In: *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016*. 2016, pp. 413–419
- F. Neumann, S. Polyakovskiy, M. Skutella, L. Stougie, and J. Wu. "A Fully Polynomial Time Approximation Scheme for Packing While Traveling". In: *CoRR* abs/1702.05217 (2017). arXiv: 1702.05217. URL: http://arxiv.org/abs/1702.05217
- J. Wu, M. Wagner, S. Polyakovskiy, and F. Neumann. "Exact Approaches for the Travelling Thief Problem". In: *Simulated Evolution and Learning - 11th International Conference, SEAL 2017, Shenzhen, China, November 10-13, 2017, Proceedings*. 2017, pp. 110–121
- J. Wu, S. Polyakovskiy, M. Wagner, and F. Neumann. "Evolutionary Computation plus Dynamic Programming for the Bi-Objective Travelling Thief Problem". In: *CoRR* abs/1802.02434 (2018). arXiv: 1802.02434. URL: http://arxiv.org/abs/1802.02434

The conclusion to the whole thesis is drawn in Chapter 9.

Chapter 2

Real-World Multi-Component Optimisation Problems and the Travelling Thief Problem

As we state in the introduction (Chapter 1), real-world *multi-component systems* are usually hard to solve. One of the most important reasons for this is that tackling such problems relies on proper corresponding models. However, most of the existing models (i.e. the benchmark problems) may have drawbacks. For example, some focus too much on individual silos, some only consider the interactions between simple test functions or homogeneous subcomponents. Among them, the travelling thief problem (TTP) combines two disparate $\mathcal{NP}$-hard components: the travelling salesman problem (TSP) and the knapsack problem (KP), which represents the real-world complexities profoundly and provides a unique benchmark problem for multi-component problems.

In this chapter, we introduce the general background of the TTP, giving the definition of the general optimisation problem in Section 2.1. In Section 2.2, we enumerate two real-world multi-component cases of optimisation, in which solving individual components cannot guarantee the overall optimality. We then introduce a newly proposed benchmark problem that is designed to imitate the real-world multi-component problems more effectively in Section 2.3.

2.1 Optimisation Problem

In mathematics and computer science, an optimisation problem is designed to locate the best solution from all feasible solutions. It is therefore commonly considered as a type of search problem, where the search space contains all potential solutions. Optimisation problems can be categorised into those which are continuous and those which are discrete, depending on the type of their variables, as well as single or multi-objective according to the number of their objective functions. In a discrete optimisation problem, variables are discrete and countable. In contrast, continuous variables include infinitely uncountable values, for example, a variable over a non-empty range of real numbers. On the other hand, a single-objective optimisation problem only involves one objective function to be optimised. In contrast, a multi-objective one considers two or more objectives simultaneously, meaning the interaction or the trade-off between objectives must be taken into account.

2.1.1 Single Objective Optimisation Problems

A standard form of a single objective continuous optimisation problem can be defined as follows:

$$\begin{aligned} &\underset{x}{minimise} && f(x) \\ &s.t. && g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & && h_j(x) = 0, \quad j = 1, \ldots, p, \end{aligned} \tag{2.1}$$

where f, g_i and h_j are general functions of the parametric n-variable vector $x \in \mathbb{R}^n$. Among them, f is the objective function to be minimised, $g_i(x) \leq 0$ are m *inequality constraints*, $h_j(x) = 0$ are q *equality constraints*, and $m \geq 0$ and $p \geq 0$. The problem is an unconstrained optimisation problem if there is no constraint being incorporated, i.e. m and p equal 0. While the form defines a minimisation problem as a convention, a maximisation problem can be treated by negating the objective function.

In some optimisation problems the variables make sense only if they take on integer values, i.e. $x \in \mathbb{Z}^n$ for (2.1), which is a type of discrete optimisation problem. Generally, discrete optimisation problems may contain not only integer variables, but also more abstract variable objects such as binaries, permutations of an ordered set or a graph. The defining feature of a discrete optimisation problem is that the variable x is drawn from a finite (but often very large) set [131].

Combinatorial optimisation is a field of applied mathematics, combining techniques from combinatorics, linear programming, and the theory of algorithms, to solve discrete optimisation problems [35], and therefore is usually used as an alias of discrete optimisation [136]. As formally defined by Neumann et al. [130], a combinatorial optimisation problem can generally be drawn as a triple (S, f, Ω), where S is a given search space, f is the objective function, which should be either maximised or minimised, and Ω is the set of constraints that have to be fulfilled to obtain feasible solutions. The goal is to find a globally optimal solution, meaning a solution $s^* \in S$, with either the highest objective value in the case of maximisation or the smallest objective value in the case of minimisation, each under the condition that all constraints are fulfilled.

Nocedal et al. [131] believe that continuous optimisation problems are normally easier to solve than their discrete counterparts, owning to the smoothness of the functions, which allows the use of calculus techniques. In general, calculus techniques mean to use objective and constraint information at a particular point x, to deduce information about the behaviour of the objective function at all points close to x. The deduced information is then used to guide the search direction. In contrast, combinatorial optimisation problems are to some extent harder to solve than the continuous ones. This is because the behaviour of the objective and constraints may change significantly as we move from one feasible point to another, even if the two points are close according to some measure [131]. Thus it is usually not possible to deduce the information of the neighbouring points from the current one.

2.1.2 Multi-objective Optimisation Problems

When incorporating more than one objective function to be optimised simultaneously, an optimisation problem becomes a multi-objective optimisation problem, which has been applied in many fields of science, including engineering, economics and logistics, where optimal decisions need to be taken in the presence of trade-offs between two or more conflicting objectives. Mathematically, in a k-objective optimisation problem, the objective function $f(x)$ is defined as follows [26]:

$$f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \cdots \\ f_k(x) \end{bmatrix},$$

where x is the parametric n-variable vector $x \in \mathbb{R}^n$ if it is a continuous optimisation problem, or a discrete structure for combinatorial optimisation.

In mathematical terms, a multi-objective optimisation problem can be formulated into a minimisation problem as follows:

$$\begin{aligned} \underset{x}{minimise} \quad & (f_1(x), f_2(x), \ldots, f_k(x)) \\ s.t. \quad & x \in X, \end{aligned} \tag{2.2}$$

where the set X is the set of feasible solutions. Similar to single-objective optimisation, the feasible set is typically defined by some constraints, and an objective function to be maximised is equivalent to minimise its negative.

In general, when dealing with a multi-objective optimisation problem, a solutions x_1 is said to dominate x_2 (denoted by $x_1 \preceq x_2$) if x_1 is better than x_2 on all objectives. Mathematically for the minimisation case, this means the following two conditions are held if $x_1 \preceq x_2$ [26]:

$$\forall i \in \{1, \ldots, k\} : \ f_i(x_1) \leq f_i(x_2), \tag{2.3}$$

$$\exists j \in \{1, \ldots, k\} : \ f_j(x_1) < f_j(x_2). \tag{2.4}$$

If x_1 does not dominate x_2 and x_2 does not dominate x_1 either, they are called non-dominated solutions.

For a nontrivial multi-objective optimisation problem, no single solution exists that simultaneously optimises each objective, namely no single solution that dominates all other solutions. Therefore the expected result of optimising such a problem is a set of non-dominated solutions, which is known as the *Pareto front* or *Pareto optimal set* [33]. In addition, without additional subjective preference information, all solutions in a Pareto front are considered equally good (as vectors cannot be ordered completely).

2.2 Real-World Multi-Component Optimisation Problems

In this section we introduce two modern real-world cases of multi-component optimisation problems, which demonstrate the complexity of such complex systems. Moreover, the second case directly inspires a new benchmark problem that is designed to better imitate the multi-component problems.

2.2.1 The Wave Energy Converters Optimisation Problem

Our first case is from the renewable energy industry. Global energy demand is on the rise nowadays, and given that there are finite reserves of fossil fuels, renewable forms of energy are playing an ever more important role in our energy supply [105]. Wave energy is a widely available but largely unexploited source of renewable energy, with the potential to make a substantial contribution to future energy production [46, 91]. The idea of harnessing wave energy has been around for at least two centuries, with the first patent for a wave energy device being filed in 1799 [54]. However, it was not until the oil crisis of the 1970s and the publication of Stephen Salter's iconic paper in Nature [150] that interest in wave energy truly began to surge. Since that time, the utilisation of wave energy has continued to be a very active research area. There are currently dozens of ongoing wave energy projects at various stages of development, exploring a variety of techniques [46, 91, 54, 103].

A device that captures and converts wave energy to electricity is often referred to as a wave energy device or wave energy converter (WEC). One common WEC design is called a point absorber or buoy. The buoy typically floats on the surface, or just below the surface, of the water and captures energy from the movement of the waves [91]. An example of a point absorber is the CETO wave energy converter[1], developed by Carnegie Wave Energy and named after the Greek sea goddess Ceto [107]. The CETO system consists of one or more fully submerged buoys that are tethered to the seabed in an offshore location, as shown in Figure 2.1. These buoys use the motion of the waves to drive a hermetically sealed hydraulic line, which in turn drives hydroelectric turbines to generate electricity, or to power a reverse osmosis desalination plant to create potable water [106].

One of the central goals in designing and operating a wave energy device is to maximise its overall energy absorption. As a result, the optimisation of various aspects of wave energy converters is an important and active area of research. Three key aspects that are often optimised are geometry, control, and positioning. Geometric optimisation seeks to improve the shape and/or dimensions of a wave energy converter (or some part of it) with the objective of maximising energy capture [115, 124]. On the other hand, the optimisation of control is concerned with finding good strategies for actively controlling a WEC [132]. A suitable control strategy is needed for achieving high WEC performance in real seas and oceans, due to the presence of irregular waves [70]. In our study we focus on the third aspect, namely the positioning of wave energy converters.

[1] https://www.carnegiece.com/wave/what-is-ceto/

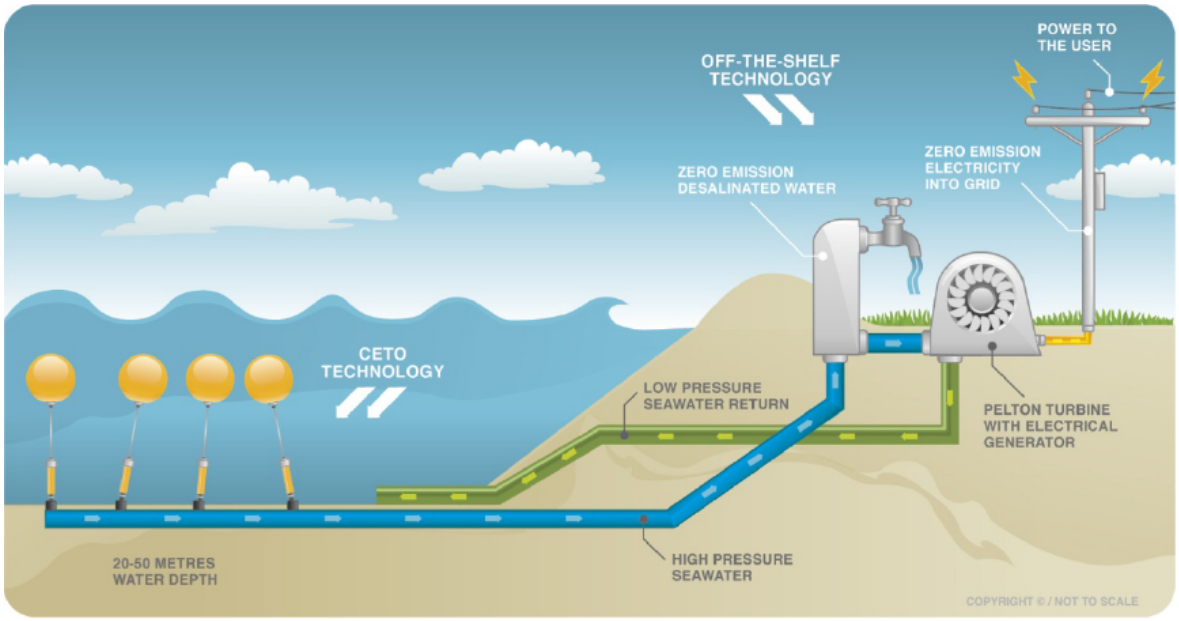


FIGURE 2.1: Operation of the CETO system [106].

A single wave energy converter can only capture a limited amount of energy. For large-scale wave energy production and in order to make any significant contribution to addressing global energy demand, it is essential to deploy wave energy devices in large numbers. A group of wave energy devices working in close proximity to one another is referred to as a wave energy farm or array [2]. Just as the optimisation of individual wave energy devices is an area of research, so is the optimisation of arrays of such devices. In the case of arrays, the aspects that are typically optimised include the layout or configuration of the array [29] and active control of individual devices [58].

Despite the fact that the size of an array is relatively small (100 WECs in the largest cases in our study), the interaction between WECs is quite complex. A simulation of such interactions across an array with 50 WECs takes around 35 hours, which is the first major obstacle for applying iterative optimisation approaches to the farm. By designing an approximation model and making numerous efforts to optimise the simulation, we eventually speed up the simulation by around 6 minutes, which enables us to do the optimisation of the overall performance of the array. The lessons we have learned from this work are two-fold. First, real-world optimisation problems can be too complex to apply iterative optimisation approaches to them, such as for local search, evolutionary algorithms, etc., due to the computational cost of evaluating the state. Second, the size of the problem is not the only source of complexity. The interactions of among individuals or components may well introduce difficulties. The details of this study will be elaborated in Chapter 4.

2.2.2 The Transportation of Water Tanks Problem

Another example is the transportation of water tanks, which was introduced by Stolk et al. [159] and Bonyadi et al. [24]. In an Australian company, tanks with a large volume are produced and delivered to geographically dispersed buyers. Due to the

high cost of transport, the decision makers of the company have to minimise the cost according to a key performance indicator (KPI) that is: to find the transport costs as a percentage of the delivered goods' value. The options that have to be taken into account include: a) selection among several production plants or dozens of storage facilities for picking available products to deliver; b) the choice of various types of trucks, with or without trailers, whether suitable or not, to transport the goods; c) packing products on trucks in an optimal way; d) selection of drivers according to their qualifications, timetables, etc.; e) selection of optimal routes to travel; and so on.

Most of the above choices are clearly hard enough to solve even in isolation. For instance, the packing problem is at least as hard as the 3-dimensional bin packing problem, which is a famous $\mathcal{NP}$-hard problem. The optimal routes to travel can be also approximated as an instance of the TSP if only one vehicle is considered, or of the VRP if a fleet of vehicles is taken into account.

However, minimising the overall KPI is even harder than making each decision separately, as the decisions made for one problem may impact the others. For example, the goods packed on a truck may influence its optimal routes, as different goods might need to be delivered to different buyers, hence different destinations. Moreover, water tanks can often be bundled inside each other to save space during the transportation period. But such bundled tanks have to be unbundled at special locations nominated by the company before being shipped to customers. Therefore, the route to the special sites has to be included when optimising the routes in this case. More commonly, different tanks loaded on a truck might yield different costs as they might require different types of trucks, drivers or petrol consumption. In other words, the sub-problems, such as the packing planning problem and the routing planning problem, are intertwined. Solving each sub-problem in isolation can not yield the overall optimality in this real-world case.

2.3 The Travelling Thief Problem

As shown in our examples, modern real-world optimisation problems are complex and often composed of multiple components or sub-problems that are classically hard. In recent years, the so-called multi-component problems have gained special interest [23, 159]. Such problems combine different optimisation problems into a single problem and are motivated by complexity issues arising in the areas such as logistics and supply chain management [87], especially for problems similar to the transportation of water tanks in the previous section.

While multi-component problems are being studied, the main question is what makes the combination of underlying problems harder to solve than solving each of the subproblems separately. Recently, the travelling thief problem (TTP) [24] has been introduced to study the interdependence of two classic optimisation problems in a systematic way and to gain better insights into the design of multi-component problems. The TTP combines the TSP and the KP by making the speed that a vehicle travels along a TSP tour dependent on the weight of the selected items. Furthermore, the overall objective is given by the sum of the profits of the collected items minus the weight dependent travel cost along the chosen route. It reflects

the complexity in real-world applications that can commonly be observed in the areas of planning, scheduling and routing. For example, delivery problems usually consist of a routing part for the vehicle(s) and a packing part of the goods onto the vehicle(s).

Bonyadi et al. [24] has given a thorough description of the TTP. Herein we revisit their definitions of the TTP and its two sub-problems here, and then briefly introduce the relevant state-of-the-art development of the TTP, including a simplified version of the TTP and existing approaches for it.

2.3.1 The Travelling Salesman Problem

The TSP is one of the most intensively studied optimisation problems in computational mathematics and was first formulated in 1930 [3]. In a nutshell, it can be defined as the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the original city?"

More formally, Letting $N = \{1, ..., n\}$ be the list of cities in the size of n, and $D = (d_{ij}) \in \mathbb{R}^{n \times n}$ denote the distance matrix in which an element d_{ij} represents the distance between city $i \in N$ and $j \in N$, we have the objective function of the TSP denoted by:

$$f(\pi) = \sum_{i=1}^{n-1} d_{x_i x_{i+1}} + d_{x_n x_1}, \quad \Big(\pi = (x_1, ..., x_n), x_i \in N\Big). \tag{2.5}$$

Here π represents a tour, which contains all of the cities in N exactly once, i.e. a permutation of N. The matrix D is defined differently according to the physical or theoretical meanings of instances. Popular types of distance metrics include Euclidean distance, geometric distance, etc. The overall goal of the TSP is to minimise the objective function f.

The well known benchmark library of the TSP is TSPlib collected by Reinelt [145]. It consists of hundreds of instances with sizes ranging from 14 to 85,900 cities, which are derived from either industrial applications or geographic problems. Since its creation in 1990, it has become the standard measurement of study about the TSP and has been extensively investigated.

2.3.2 The Knapsack Problem

The KP is another classic $\mathcal{NP}$-hard problem. Given a set of items, each with a weight and a profit, a subset is to be determined, from which the total profit is maximised while its total weight is under a limit (i.e. capacity). Formally, letting $M = \{1, ..., m\}$ be the set of items, p_i and w_i denote the profit and the weight respectively of item

$i \in M$, and C be the capacity, we have the definition of a KP as follows:

$$maximise \quad f(\rho) = \sum_{i=1}^{n} p_i y_i,$$
$$s.t. \quad \sum_{i=1}^{n} w_i y_i \leq C.$$

Here $\rho = (y_1, ..., y_m)$, $y_i \in \{0, 1\}$ represents the packing plan of the item set, meaning the item i is chosen when $y_i = 1$, and vice versa. As y_i is binary, this problem is so called the 0-1 knapsack problem (KP).

The KP has been thoroughly studied by Martello, Pisinger and Toth [111, 110, 165, 138, 112, 109]. Their work mainly concentrates on proposing the exact algorithms to tackle the 0-1 KP. In order to test their algorithms, they propose a KP benchmark library that is generated in fifteen different types. The library contains the KP instances, varying in weights and profits from items. The authors use it to investigate which aspects result in challenges for KP solvers in general and for their approach in particular.

2.3.3 The Definition of the TTP

In order to combine the above two problems together, Bonyadi et al. [24] firstly assume that a set of cities $N = \{1, \ldots, n\}$ and a set of items $M = \{1, \ldots, m\}$ are given. City i, $i = 2, \ldots, n$, contains a set of items $M_i = \{1, \ldots, m_i\}$, $M = \bigcup_{i \in N} M_i$. Item k positioned in the city i is characterised by its profit p_{ik} and weight w_{ik}. A thief must visit each of the cities exactly once and then return back to the starting city at the end. The distance matrix $D = \{d_{ij}\}$ for every pair of cities $i, j \in N$ is known. Any item may be selected, as long as the total weight of the collected items does not exceed the capacity C, represented by a packing plan $\rho = (y_{21}, \ldots, y_{nm_n})$, $y_{ik} \in \{0, 1\}$.

They then introduce a new parameter - the velocity of the thief travelling along a tour $\pi = (x_1, \ldots, x_n)$, $x_i \in N$, defined as follows:

$$v_{x_i} = v_{max} - \nu W_{x_i}. \tag{2.6}$$

Here $v_{x_i} \in [v_{min}, v_{max}]$ denotes the velocity when the thief leaves from city x_i. v_{min} and v_{max} denotes the maximal and minimal speeds of the thief, which are the predefined constants. ν is a coefficient, defined by $\nu = (v_{max} - v_{min}) / C$. W_{x_i} is the accumulated weight of the items collected in the preceding cities $x_1, \ldots, x_i$, computed by:

$$W_{x_i} = \sum_{j=1}^{i} \sum_{k=1}^{m_{x_j}} w_{x_j k} y_{x_j k}. \tag{2.7}$$

The overall travelling time $\mathcal{T}(\pi, \rho)$ can therefore be calculated as follows:

$$\mathcal{T}(\pi, \rho) = \frac{d_{x_n x_1}}{v_{x_n}} + \sum_{i=1}^{n-1} \frac{d_{x_i x_{i+1}}}{v_{x_i}}. \tag{2.8}$$

Consequently, the accumulated weights have a negative impact on the overall travelling time $\mathcal{T}(\pi, \rho)$, which introduces the nonlinearity. The time increases when more items are carried.

Given the total profit of the items, $\mathcal{P}(\rho)$ is computed by:

$$\mathcal{P}(\rho) = \sum_{j=1}^{n} \sum_{k=1}^{m_{x_j}} p_{x_j k} y_{x_j k}, \tag{2.9}$$

we can calculate the overall reward by:

$$\mathcal{Z}(\pi, \rho) = \mathcal{P}(\rho) - R \cdot \mathcal{T}(\pi, \rho). \tag{2.10}$$

Here R is an constant, introduced to connect the total profit and the overall travelling time. It is semantically a renting rate that is to be paid for each time unit taken to complete the tour.

The minuend $R \cdot \mathcal{T}(\pi, \rho)$ therefore becomes the *total transportation cost*, which non-linearly depends on the weight of collected items.

$\mathcal{Z}(\pi, \rho)$ can be expanded as follows.

$$\mathcal{Z}(\pi, \rho) = \sum_{i=1}^{n} \sum_{k=1}^{m_i} p_{x_i k} y_{x_i k} - R \cdot \left(\frac{d_{x_n x_1}}{v_{max} - \nu W_{x_n}} + \sum_{i=1}^{n-1} \frac{d_{x_i x_{i+1}}}{v_{max} - \nu W_{x_i}} \right) \tag{2.11}$$

The sole objective of this problem is to find the optimal total reward $\max_{\pi,\rho} \mathcal{Z}(\pi, \rho)$, which is the so-called Model I of the TTP.

Correspondingly, there is another model in [24], i.e. Model II, which considers two objectives, namely maximising the total value while minimising the travel time. The total value here is the value of all the picked items, after completing the tour, which is measured according to a dropping rate $Dr^{\lceil \frac{T_i}{c} \rceil}$ for each item. Here T_i is the total time that the item i is carried, and c is a constant. The value of item i therefore is $p_i \cdot Dr^{\lceil \frac{T_i}{c} \rceil}$.

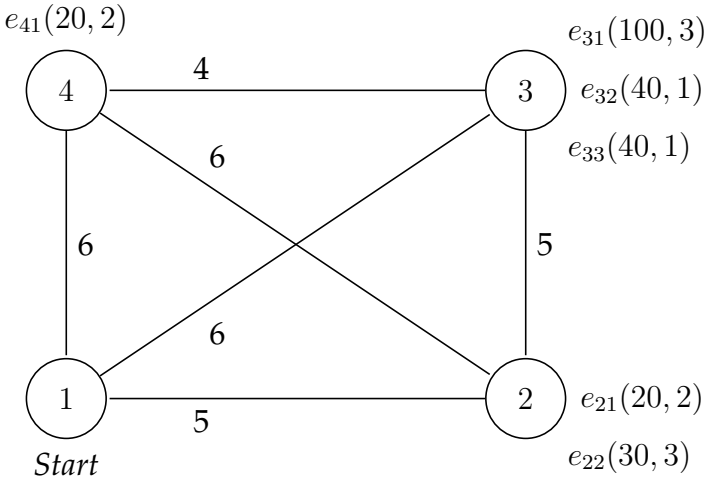


FIGURE 2.2: An Illustrative Example [24, 139]

A very simple example of the TTP problem with 4 cities is provided, as shown in Figure 2.2 [24, 139]. Each city, except the starting city 1, has been assigned a set of items, e.g. city 2 has item e_{21} of profit $p_{21} = 20$ and weight $w_{21} = 2$, and item e_{22} of profit $p_{22} = 30$ and weight $w_{22} = 3$. We assume that the capacity is $C = 3$, the renting rate $R = 1$ and v_{max} and v_{min} are set as 1 and 0.1, respectively. Then the objective value $\mathcal{Z}(\pi, \rho) = 50$ for $\pi = (1, 2, 4, 3)$ and $\rho = (0, 0, 0, 1, 1, 0)$. More specifically, the overall reward from city 1 to city 4 is -15, as the thief does not collect any items while travelling through the first 3 cities. In the city 3, two items, e_{32} and e_{33}, are picked up, which gives a total profit of 80 and a carried weight of 2. On the final journey from city 3 back to city 1, the carried weight reduces the speed and results in increased costs of 15. Consequently, the final objective value is $\mathcal{Z}(\pi, \rho) = -15 + 80 - 15 = 50$.

2.3.4 The Packing While Travelling Problem

The packing while travelling problem (PWT) has been introduced in [140] to push forward theoretical studies on the TTP. It concentrates on the packing part of the TTP, as this part can be regarded as a combination of a classical 0-1 knapsack problem with a non-linear travel cost function. The study of non-linear planning problems is an important topic and the design of approximation algorithms has gained increasing interest in recent years [81, 179].

The PWT can be seen as the TTP when the route is fixed but the cost still depends on the weight of the items on the vehicle. The problem is motivated by gaining advanced precision when minimising transportation costs that may have a non-linear nature, for example, in applications where weight impacts the fuel costs [66, 100]. From this point of view, the PWT is a baseline problem in various vehicle routing problems with non-linear costs. Some specific applications of the PWT may deal with a single truck collecting goods in large remote areas without alternative routes, that is, a single main route that a vehicle has to follow may exist while any deviations from it in order to visit particular cities are negligible [140]. The problem is $\mathcal{NP}$-hard even without the capacity constraint usually imposed on the knapsack. Furthermore, exact and approximative mixed integer programming approaches, as well as a branch-infer-and-bound approach [140] have been developed for this problem.

The PWT can be formally stated as follows. Given are a set of cities $N = \{1, \ldots, n+1\}$ and a set M of m items distributed among the first n cities. Thus, each city i, $1 \leq i \leq n$, has a set of items $M_i \subseteq M$, $|M_i| = m_i$. Each item $e_{ik} \in M_i$ is characterised by a positive integer profit $p_{ik} \in [P_L, P_U]$ and a positive integer weight $w_{ik} \in [W_L, W_U]$, where P_L, P_U and W_L, W_U are the lower and upper bounds on p_{ik} and w_{ik}, respectively. There is a predefined tour $\pi = (x_1, \ldots, x_{n+1})$, $x_i \in N$ given to a thief, who may collect any items, so that each of the cities must be visited in the fixed order exactly once.

Let $y_{ik} \in \{0, 1\}$ be a variable indicating whether or not item $e_{ik} \in M$ is chosen in a solution. Then a packing plan $\rho = (y_{11}, \ldots, y_{nm_n})$ represents a solution to the PWT problem. It is notable that the essence of the solution or the packing plan ρ is a set of selected items, i.e. $\{e_{ik} | e_{ik} \in M,\ y_{ik} = 1\}$. The thief is travelling between the cities in a rented vehicle with a variable velocity. When travelling from city i to $i + 1$, it

has velocity $v_i \in [v_{min}, v_{max}]$ which depends on the weight of the items collected in the first i cities,

$$v_i = v_{max} - \nu \sum_{j=1}^{i} \sum_{e_{jk} \in M_j} w_{jk} y_{jk}, \tag{2.12}$$

where ν is a coefficient and C is the capacity of the vehicle, the same as in the TTP. In addition, the PWT problem is unconstrained iff the total weight of all items in M is less than or equal to C. In practice, we can assume that capacity C is always equal to $m \times W_U$ in order to keep the problem unconstrained when we focus on the investigation of the unconstrained version.

The total reward with respect to π is calculated as

$$\mathcal{Z}_\pi(\rho) = \mathcal{P}(\rho) - R \cdot \mathcal{T}_\pi(\rho), \tag{2.13}$$

where $\mathcal{P}(\rho)$ is the total profit of selected items and $\mathcal{T}_\pi(\rho)$ is a total travelling time. $\mathcal{P}(\rho)$ is to be computed as follows:

$$\mathcal{P}(\rho) = \sum_{i=1}^{n} \sum_{e_{ik} \in M_i} p_{ik} y_{ik} \tag{2.14}$$

As the tour π is given, determining the full distance matrix D of $\mathbb{R}^{(n+1)\times(n+1)}$ is not necessary. We consequently use a vector d of $\mathbb{R}^n$ to denote the distances between each pair of cities along the tour π. Each entry d_i, $i \in [1, n]$ represents the distance between the i-th and $i+1$-th cities on the tour. Therefore $\mathcal{T}_\pi(\rho)$ can be defined as follows:

$$\mathcal{T}_\pi(\rho) = \sum_{i=1}^{n} \frac{d_i}{v_i}. \tag{2.15}$$

Because the velocity depends on the weight of the chosen items, the total travel time increases when the thief collects more items. Given a non-negative renting rate R, $R \times \mathcal{T}_\pi(\rho)$ is the total transportation cost of the items in ρ, which contains the non-linearity, the same as in the TTP. The objective of the problem is to maximise the overall benefit $\mathcal{Z}_\pi(\rho)$.

2.3.5 The Benchmark Library of the TTP

In order to provide a unified platform for testing and comparing different approaches to the TTP, Polyakovskiy et al. [139] propose a brand new set of instances that combines the well-known test suites from each sub-problem. It thus far has been considered the main benchmark library of the TTP. The design of the library mainly tackle the challenge of balancing the two sub-problems, which means one shall not dominate another in the TTP. For example, the optimality of the TSP sub-problem shall not make the KP packing aspect negligible. Vice versa, the most profitable loading plan must not reduce the importance of a shorter tour. This requires the constants of the TTP to be well adjusted, which is not a trivial task. Chapter 5 formally illustrates

the non-triviality and elaborates the methods of obtaining harder/easier instances via adjusting a most important constant - the Renting rate R.

To generate the TTP instances, Polyakovskiy et al. [139] select 81 TSP instances from the TSPlib [145], which contains instances with the city size varied from 51 to $85,900$. In addition, they assume that all distances in the TTP library are integers. To achieve this, they round all the distances in the 81 TSP instances up to the next integer. For each TSP instance, they generate a knapsack component using three ways from Pisinger [138] and Martello et al. [108]. Each way therefore results in a unique KP type, which is *uncorrelated*, *uncorrelated with similar weights* and *bounded strongly correlated*.

According to the analysis in [108], the three types of KP represent different hardnesses for solving. The uncorrelated type having the uniformly-at-random integers for their weights and profits are easy to solve to optimality, even for large-sized instances of KP. The uncorrelated type with similar weights has employed the profits uniformly distributed within a normal range, i.e. $[1, 10^3]$, but the weights within a much narrower range, i.e. $[10^3, 10^3 + 10]$. It has already been shown that this type of 0-1 knapsack instance is more time consuming to solve [108]. Strongly correlated problems are typically very difficult. They have a normal range for the uniformly distributed weights, but each profit is strictly equals to the corresponding weight plus 100. This setting significantly increases the computational cost for the typical solvers of the KP.

Moreover, the total number of items and the capacity of the knapsack are diversified as well. In the library, items are evenly distributed in the cities from 2 to n according to a factor that describes how many items per city are available. A total 4 factors are proposed, i.e. $\{1, 3, 5, 10\}$. On the other hand, 10 categories are created to generated instances with different capacities from relatively small to large. They are generated according to the total weight of all items W multiplying a factor $c \in \{\frac{1}{11}, \ldots, \frac{10}{11}\}$.

In order to balance the two sub-problems, Polyakovskiy et al. [139] adjust the Renting rate to be $R = \frac{\mathcal{P}(\rho^*)}{\mathcal{T}(\pi^*, \rho^*)}$, where ρ^* denotes the packing plan obtained by solving the KP component solely, and π^* is the near-optimal tour obtained via the Chained Lin-Kernighan heuristic [101]. Such selection of R guarantees the existence of at least one TTP solution with a zero objective value.

Overall, the library based on 81 TSP instances, 3 KP types, 4 item factors and 10 capacity categories contains $81 \times 3 \times 4 \times 10 = 9,720$ different TTP instances in total.[2]

2.3.6 Single Objective and Multi-Objective

According to the definition in [24], there are two versions of the TTP, namely Models I and II, which are single objective and bi-objective respectively. Thus far more attention has been paid to the former [139, 25, 118, 116, 117, 168, 169, 161, 50, 56], than the latter [20, 178]. The reasons behind this might be two-fold: first, Model I has a more clearly defined objective function and the existing library [139] does not contain instances for Model II; second, since the main purpose of the TTP is focused on investigating the interdependence of the two sub-problems, it does not seem very

[2] All instances and implementations of the objective function are available online: `http://cs.adelaide.edu.au/~optlog/research/ttp.php`

necessary to examine bi-objectives before researchers understand the nature of the interdependence well, as it might introduce unnecessary complexity. In this thesis, we also mainly focus on the single objective. Nevertheless, we also investigate a new bi-objective version of the TTP in Chapter 8, which is derived from our investigation of the single objective version. We regard it as the extension of our study of Model I, instead of Model II.

2.4 Conclusion

In this chapter, we have given a brief background introduction about the multi-component problems. We have reviewed the definitions of general optimisation problems and then exemplified the multi-component optimisation problems by using two real-world case studies: the wave energy converters (WECs) optimisation problem and the transportation of water tanks problem. Inspired by the transportation of water tanks problem and other logistical optimisation problems, the travelling thief problem (TTP) was proposed as a benchmark problem of multi-component optimisation problems. We thus revisited the formal definition and the settings of the TTP as well as introduced state-of-the-art developments of it, including a simplified version: the packing while travelling (PWT) problem, and the benchmark library of the TTP. The content of this chapter defines the target problems of our studies in this thesis. In the next chapter, we will introduce general approaches for optimisation problems and exemplify some popular algorithms.

Chapter 3

Exact and Heuristic Approaches for Optimisation Problems

As introduced in the previous chapters (Chapters 1, 2), lots of optimisation problems have been thoroughly studied by researchers in the fields of applied mathematics, operation research, theoretical computer science etc. Famous examples include the travelling salesman problem (TSP) [34], knapsack problem (KP) [111], vehicle routing problem (VRP) [95], and so on. In general, most of optimisation problems are categorised as $\mathcal{NP}$-hard [98], meaning there is no existing algorithm that can solve the worst cases of any of the problems in polynomial time on a deterministic Turing machine [156], unless $\mathcal{NP} = \mathcal{P}$. In other words, they are intractable [59].

Despite their intractability, many algorithms have been proposed to tackle the problems. Among them exact approaches such as the dynamic programming (DP) [14], branch and bound search (BnB) [125], constraint programming (CP) [114], etc. have been studied widely, as such methods theoretically guarantee to yield optimality. Moreover, exact approaches might bring valuable insights into the problems. For example, dynamic programming solves the 0-1 knapsack problem very well (in pseudo-polynomial time), which suggests that the KP is composed of numerous sub-problems that contains duplicated calculations. Such insights provide better understanding of the problems and might be able to be utilised by other approaches. However, due to the nature of the $\mathcal{NP}$-hardness of the problems, exact approaches are normally bound to the problem instances with small sizes. Taking the TSP as an example, the original BnB and LP can only solve the instances containing up to around 50 and 200 cities respectively. In order to surpass this, some techniques, such as the cutting-plane method [40] etc., are introduced to improve the capability of the exact approaches. Nevertheless, the computational cost is still immense. For example, in 2006 a TSP instance with $85,900$ cities was solved using an LP solver named Concorde [4], which took over 136 CPU-years [3]. Moreover, almost all of such methods are problem-specific, meaning that such improvements for the TSP cannot be transferred to other problems easily.

On the other hand, heuristic approaches provide an alternative for dealing with the problems. These methods trade optimality, completeness, accuracy and/or precision for computational efficiency. Some of them can solve the problem instances with a size in the millions in a reasonable time, while the solutions are just 2-3% away from optimality in a high probability case [144]. Although some heuristics are still problem-specific, e.g. Lin-Kernighan [101], others, such as simulated annealing [86], tabu search [64], genetic algorithms [123], ant colony [43] etc., are mainly problem independent. They usually require minor modification for adapting the

algorithms from one problem to another, hence being so-called *meta-heuristics* [22].

In spite of the significant differences between the exact and heuristic approaches, each of them can be recognised generally as a *search algorithm* with a special strategy for the solution space of a problem. For instance, the DP is a search algorithm where the strategy is to cache and reuse the results of the branches that have been searched before; and heuristics normally explore the further areas of the search space, based on an advance evaluation of such areas. In practice, there is no clear answer about which type of approach is better than another, without taking the particular instances of an optimisation problem into consideration. According to the *no free lunch theorem* by Wolpert et al. [171], a general-purpose universal optimisation strategy is impossible, and one search strategy can outperform another if and only if it is specialised to the structure of the specific problem. This implies a ubiquitous circumstance that some particular instances of a problem might be solved well by one strategy and others might correspond to another approach. Therefore, the combination of approaches (namely hybrid approaches) are naturally desired, as such a design might bring better performance in general. In this thesis, we consider the both the exact and the heuristic, as well as hybrid approaches towards investigating them in terms of the complex and multi-component optimisation problems introduced in Chapter 2.

The chapter includes definitions of these approaches. We first describe general search algorithms in Section 3.1. Then we enumerate some exact approaches and heuristics in Sections 3.2 and 3.3 respectively. Moreover, we give an introduction to multi-objective evolutionary algorithms in Section 3.4.

3.1 Search Algorithms

Search algorithms are ubiquitous in computer science, especially for solving optimisation problems. Classically, a search algorithm is used to retrieve information stored within an existing data structure, such as an array, table, tree etc. In contrast, the search algorithms for an optimisation problem deal with the problem space that is defined by the objective function(s) and constraints, where the goal is find a variable assignment that will maximise or minimise the function(s) of those variables under the constraints. Such a space is commonly either infinite (in continuous optimisation problems) or extremely large (in combinatorial optimisation problems), as introduced in Section 2.1.

Algorithms for these problems include the very basic and general brute-force search (also called *naïve* or *exhaustive* search), which systematically enumerates all possible candidate solutions for the problem and finds the optimality among them. Although a brute-force search is simple to implement, will always find a solution if it exists, and guarantees that the found solution is optimal, it is not efficient in practice. Its cost is proportional to the number of candidate solutions, meaning it cannot handle the phenomenon so-called *combinatorial explosion* of the intractable problems, i.e. the amount of candidates explodes along with the increasing size of the problem. This implies that the search can only work on the problems with very limited size. Nevertheless, the search is useful as a baseline method when benchmarking other algorithms. Our study in Chapter 4 demonstrates such a case.

Algorithm 1 A Generic Graph Search Algorithm [149]

```
function GRAPH-SEARCH(a problem instance) return a solution, or failure
    initialise the frontier using the initial state of problem
    initialise the explored set to be empty
    loop
        if the frontier is empty then
            return failure
        end if
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then
            return the corresponding solution
        end if
        add the node to the explored set
        expand the chosen node, adding the resulting nodes to the frontier only if
not in the frontier or explored set
    end loop
end function
```

Considering the state space of a combinatorial optimisation problem is a graph, namely an ordered pair $G = (V, E)$ comprising a set V of nodes (also called vertices or points) together with a set E of edges (also called arcs or lines), where an edge $e \in E \subseteq V^2$ is the association and takes the form of the unordered pair of the vertices. Algorithm 1 demonstrates a generic approach that exhaustively explores the space, in which the *frontier* is the set of all leaf nodes (i.e. nodes without children) in the graph that are available for expansion at any given point [149].

In Algorithm 1, extra effort has been incorporated to deal with the circular paths in the graph, which introduces extra cost. Therefore, if we are dealing with the search space without circuits, such as a *tree*, the algorithm could be more efficient. Besides, some improvements can be applied to the search algorithm, for example, avoiding repetitively exploring the branches that have been expanded or pruning the branches where the goal cannot exist. The search algorithms applying such strategies may be able to improve performance, while the final optimality will still be guaranteed simultaneously, thus, this is called an *exact approach*. On the other hand, many *heuristic approaches*, which rank the branches to be explored according to certain known information and only expand search to the more promising branches, are not be able to secure optimality but they usually do provide better performance.

Nevertheless, there is no insuperable barrier between the two approaches. Indeed, a combination of approaches (namely hybrid approaches) is usually desired as such a design may bring better performance in general. In addition, the exact approaches can normally be transformed to the corresponding approximate form via relaxation techniques, for example.

3.2 Exact Approaches

In this section, we introduce the background of the three exact approaches that are used in our study. Among them, we pay special attention to dynamic programming (DP). We not only design the exact algorithms for the travelling thief problem through this approach, but also contribute some insights and a hybrid approach into the TTP. Meanwhile, the branch and bound (BnB) method and constraint programming (CP) provide the alternatives to the TTP for comparison.

3.2.1 Dynamic Programming

For a problem that can be divided into a number of sub-problems, the divide-and-conquer method [36] is a common approach. A typical divide-and-conquer approach recursively applies the following three steps to the problem:

- **Divide** the problem into a number of disjointed sub-problems;
- **Conquer** the sub-problems by solving them recursively, or in a straightforward manner if the sub-problem is simple enough;
- **Combine** the solutions to the sub-problems into the final solution for the original problem.

Dynamic programming [36] enhances the divide-and-conquer method. It applies when the sub-problems overlap, meaning the sub-problems share certain "sub-sub-problems". In this context, a divide-and-conquer algorithm does more work than necessary, i.e. repeatedly solving the common sub-sub-problems. A dynamic programming algorithm solves each sub-sub-problem just once and then stores its solution, thereby avoiding the work of re-computing the answer every time.

As an example, a naïve approach for the 0-1 knapsack problem (KP, defined in Section 2.3.2), which has an item set M and the capacity C of the knapsack, can be defined as follows:

1: **for all** subset $K \subseteq M$ **do**
2: **if** the profit of K is the greatest so far **and** the weight of K is less than or equal to C **then**
3: store K
4: **end if**
5: **end for**
6: **return** the best K

where the procedure of dealing with a subset K (namely lines 2 and 3) can be regarded as a sub-problem. The overall complexity of this algorithm will be $\mathcal{O}(2^m)$. However, it is not necessary to repetitively compute every subset of items, as every subset K' can be regarded as a new item i adding to an existing subset K, where $K = K' \setminus \{i\}$. This means that if we store the intermediate results of K and calculate K' based on K, the complexity will be reduced accordingly, which is the essence of dynamic programming. Moreover, for some subsets with an identical weight, we only need to store the best result for the weight instead of for every subset. As the weights of items are integers, this lightens the complexity to be the *pseudo-polynomial time*, which is bound by $\mathcal{O}(mC)$.

Algorithm 2 Dynamic Programming for 0-1 Knapsack Problem

```
function DP-KNAPSACK(M, C)
    for w = 0 to C do
        T[0, w] = 0
    end for
    for i = 1 to |M| do
        for w = 0 to C do
            if w_i ≤ w then
                T[i, w] = max(T[i − 1, w], p_i + T[i − 1, w − w_i])
            else
                T[i, w] = T[i − 1, w]
            end if
        end for
    end for
    return T[n, C]
end function
```

Algorithm 2 sketches this approach, where the intermediate results are stored in a two-dimensional table T and w_i and p_i are the weight and profit of item i respectively. The key part of this algorithm is the recursive step:

$$T[i, w] = \max(T[i-1, w], p_i + T[i-1, w-w_i]),$$

which means that when we are deciding to add a new item i into a subset K, we only have two choices: either taking it or leaving it, depending on the maximal gain we can have. In Chapters 6 and 7, we will introduce similar but more complex strategies designed for the travelling thief problem.

The dynamic programming for a 0-1 knapsack problem can be relaxed to a fully polynomial time approximation (FPTAS) [84], which can produce a solution within $1-\epsilon$ of being optimal in polynomial in both the problem size m and $\frac{1}{\epsilon}$, given a parameter $\epsilon > 0$. In Chapter 6 we introduce a FPTAS for the parking while travelling problem.

3.2.2 The Branch and Bound Method

Branch and bound (BnB) is another type of search strategy for optimisation problems. The approaches normally consider the search space as a rooted tree and systematically explores the branches of this tree. Before enumerating the candidate solutions of a branch, the branch is checked against upper and lower estimated bounds on the optimal solution, and is discarded if it is verified to be unable to produce a better solution than the best one found so far by the algorithm. This method thus avoids spending unnecessary effort on such branches. In general, a BnB algorithm operates according to two principles:

- It recursively splits the search space (such as a tree) into smaller spaces (namely branches), then optimises the objective function $f(x)$ on these smaller spaces. The splitting is called *branching*, which is usually exhaustive.

Algorithm 3 A Generic Branch and Bound Method for Minimising a Function f [32]

initialise the bound B.
initialise a queue to hold a partial solution with none of the variables of the problem assigned.
repeat
  take a node N off the queue.
  if N represents a full solution x **and** $f(x) < B$ **then**
    x is the best solution so far, record it and set $B = f(x)$.
  else
    while there is N_i produced from N **do**
      if $g(N_i) > B$ **then**
        discard N_i
      else
        store N_i on the queue
      end if
    end while
  end if
until the queue is empty

- To improve on the performance, the BnB algorithm keeps track of the best bounds, and uses these bounds to "prune" the search space, eliminating the smaller spaces that can be proved not to contain an optimal solution. This action is called *bounding*.

Algorithm 3 is the skeleton of a generic branch and bound algorithm for minimising an arbitrary objective function f [32]. In practice, the initial value of the bound B is commonly created by letting $B = f(x_h)$, where x_h is a solution generated by an efficient and approximate approach. The bound can also be initialised infinity for minimising problems if no such approach exists. The function g is a bounding function that computes the lower bound of the partial solution N_i. It has usually been defined according to the specific problem. In Chapter 7, we define a bounding function for the travelling thief problem.

3.2.3 Constraint Programming

Constraint programming (CP) is a programming paradigm wherein relationships between variables are stated in the form of constraints, i.e. representing the problem to be solved into a constraint satisfaction problem (CSP) [149]. A CSP can be represented as a triple $\mathcal{P} = (X, D, C)$, where X is an n-tuple of variables $X = (x_1, x_2, \ldots, x_n)$, D is a n-tuple of domains $D = (D_1, D_2, \ldots, Dn)$ correspondingly, and each $x_i \in D_i$, C is a t-tuple of constraints $C = (C_1, C_2, \ldots, C_t)$. A constraint C_i is a pair (S_i, R_{S_i}), where S_i is a subset of variables in X and R_{S_i} is a relation on the variables in S_i. The relation R_{S_i} can be defined either by enumerating every value that satisfies the constraint or an abstract one such as an expression of the variables in S_i.

As constraint programming focuses more on formulating the properties of a solution to be found instead of specifying a sequence of steps to execute in order to

find a solution, it is called a form of "declarative programming", which differs from the common primitives of "imperative programming" languages. In practice, people normally focus on representing their problems to be solved into the CSP expressions, and using existing software libraries, such as IBM ILOG CP Optimiser, to tackle them.

The solving techniques used in processing the CSP usually consist of two portions: 1) an exhaustive search technique for systematic exploration of the space of solutions (even though the approximate approaches, such as local search, can also be used as solvers, we do not consider them here as they can not guarantee the optimality). The search named *backtracking* is often used in this part. 2) the techniques that can improve the efficiency of the exhaustive search, such as consistency techniques, network structure based techniques, and domain-specific techniques. As the details of such techniques are beyond the scope of this thesis, we refer the interested reader to the texts by Russell et al. [149], Rossi et al. [146], Hooker [77], etc.

3.3 Heuristics

A heuristic search, also known as a *informed search*, is a search strategy that uses the additional knowledge of the problem to guide or prioritise the search path [149]. In this way, the solutions can be found more efficiently, but with no guarantee of optimality in most cases.

For example, in terms of the generic search algorithm in Algorithm 1, we apply a slight modification to it in which we sort the nodes to be expanded according to an evaluation function f and choose the node n with minimal value of $f(n)$ to be expanded first. This strategy is called *best-first search*. Moreover, if we have a *heuristic function* $h(n)$ that can estimate the cost of the path from the current node n to the final goal node according to some knowledge or experience, we can use this function $h(n)$ as the evaluation, i.e. $f(n) = h(n)$. This strategy is one of the classical heuristic searches, named a *greedy* or *best-first search*. In general, a heuristic search depends on a heuristic function that is mainly a *rule of thumb* to guide the search path in order to improve efficiency.

As heuristics do not guarantee optimality, they can be called *approximate approaches*. However, they are different from the approximation algorithms that are able to provide a bound on the quality of the returned solutions, such as the fully polynomial-time approximation scheme (FPTAS). In general, heuristics do not formulate such a bound, unless with special design.

In addition, there are two types of heuristics: *construction* and *improvement*. The former constructs one solution from scratch by performing iterative construction steps where parts of the solution are determined in each step and the process stops after the solution is complete. The latter follows a few search strategies to iteratively improves the new, complete solution(s) from inferior to superior one(s) [147].

Traditionally, heuristics are problem-specific. However, modern heuristics have been developed to be problem-independent, based on the improvement mechanism, these are defined as *meta-heuristics*. They are usually applicable to a large variety of problems with known, very limited, problem specifications. Meta-heuristics follow a process with two phases, namely *intensification* and *diversification*, which are performed alternately in the search period. The heuristics focus on exploiting the

promising areas of the search space during the intensification phase, as well as exploring new areas in the diversification phase [147].

Moreover, heuristics can be divided into single-solution methods and population-based ones. The former only generate, maintain and improve one current incumbent solution in every iteration. Examples include local search (LS), simulated annealing (SA), etc. In contrast, approaches such as evolutionary algorithms (EA) and memetic algorithm (MA) usually manipulate a set of solutions in each generation, thus they are called population-based heuristics.

Heuristics have been attractive for operations research and artificial intelligence for decades [149, 76, 147, 65, 90, 122, 8, 61, 63, 142, 154, 21, 130, 48, 119]. In this section, we introduce some common heuristics for optimisation problems, especially for the travelling thief problem, most of which can be defined as meta-heuristics.

3.3.1 Local Search

Local search (LS) is a series of heuristics that focuses on searching the neighbourhood of current state n. In some literature, it is called the "greedy heuristic" [162]. The capabilities of its heuristic function are mainly two-fold: 1) finding all neighbours of the current state according to predefined distances between the states. 2) selecting a state among the neighbours as the next current state according to certain selection rules.

The search is not systematic, comparing with other searches like an exhaustive search or best-first search. However, in systematic searches, the algorithms have to include a strategy that prevents the search going back to visited paths yet covers all the paths that have not been visited (such as the frontier shown in Algorithm 1 of Section 3.1). This strategy requires extra memory, with costs potentially being significantly high. Local searches do not have this drawback, as they only focus on the current state. Therefore, the memory cost of local searches is very low and usually a constant amount. Moreover, a local search can often find reasonable solutions in the infinite or vast search space, where a systematic counterpart commonly fails.

Algorithm 4 depicts a basic local search called a *hill-climbing* (also known as a steepest-ascent) search algorithm [149], which is a simple loop that continually moves towards the "uphill", which is the direction of those neighbours with higher

Algorithm 4 A Generic Hill-Climbing Search [149]

```
function HILL-CLIMBING(problem) return a state that is a local maximum
    current = Make-Node(problem.Initial-State)
    loop
        neighbour = a highest-valued successor of current
        if neighbour.Value ≤ current.Value then
            return current.State
        end if
        current = neighbour
    end loop
end function
```

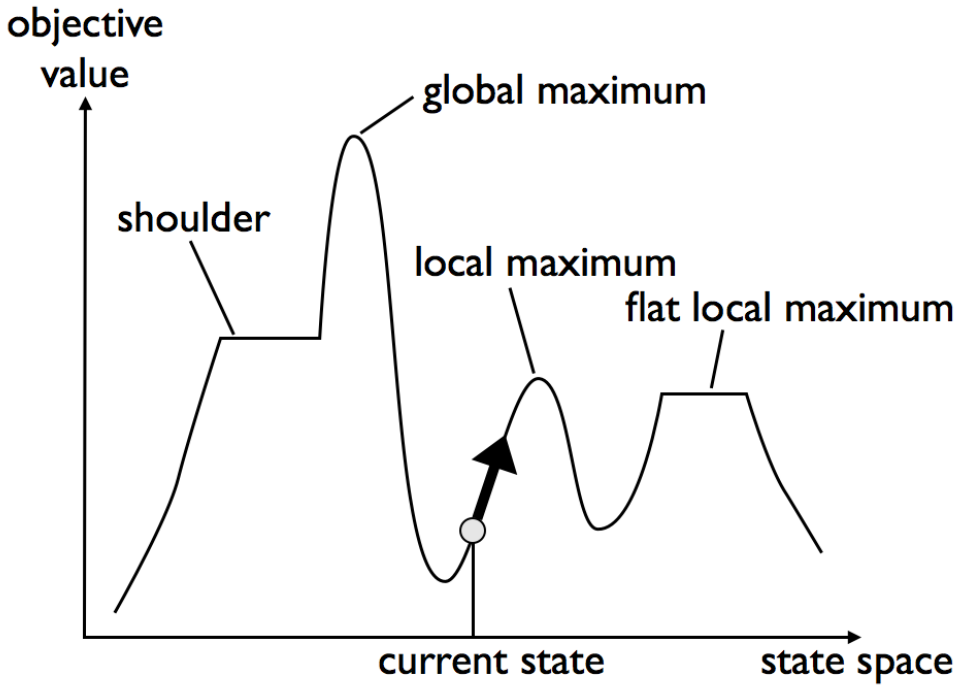


FIGURE 3.1: A one-dimensional state-space landscape in which elevation corresponds to the objective value. The aim is to find the global maximum. Hill-climbing search modifies the current state to try to improve it, as shown by the arrow. [149]

values. It terminates when it reaches a "peak", where no neighbour has a higher value than the current one.

However, the success of hill climbing depends very much on the shape of the state space, or *landscape*. In a landscape that is full of local maxima and plateaux, the algorithm is easily stuck in a "peak" that is not globally optimal. As shown in Figure 3.1 [149], when the algorithm starts at the circled point, it will inevitably terminate at the local maximum instead of the global maximum. In order to overcome this drawback, some enhancements of the local search, such as simulate annealing (SA) and genetic algorithms (GAs), incorporate additional strategies in order to escape from the local optima.

3.3.2 Simulated Annealing

Simulated annealing (SA) is a representative example of meta-heuristics. It was proposed by Kirkpatrick et al. [86] and Černý [27] independently, and can be used for combinatorial as well as continuous optimisation problems.

The name and inspiration for SA come from metallurgy, where annealing denotes a technique involving heating and properly controlled cooling of a metal to increase the size of its crystals and reduce their defects. The SA introduces the *random walk* into hill climbing in a way similar to annealing. A random walk is a move to a successor chosen uniformly at random from the set of successors. In contrast

Algorithm 5 A Generic Simulated Annealing Heuristic [149]

function SIMULATED-ANNEALING(*problem*, *schedule*) **return** a solution
  x = Make-Node(*problem*.Initial-State)
  for $t = 1$ **to** ∞ **do**
    $T = schedule(t)$
    if $T = 0$ **then**
      return x
    end if
    $x' =$ a randomly selected successor of x
    $\Delta E = f(x') - f(x)$
    if $\Delta E > 0$ **then**
      $x = x'$
    else
      $x = x'$ only with probability $e^{\frac{\Delta E}{T}}$
    end if
  end for
end function

with the hill climbing, it does not guarantee the solutions in the next iteration are better than the current ones. However, it introduces diversification, thereby widening the range of exploration. In the heuristic, the random walk is designed as having a high probability of being performed at the beginning of the run. This probability decreases in a controlled way, similar to the cooling process. This design allows the search to accept worse solutions and therefore encourages the run to escape from local optima in the early stages. During the later period, the probability of moving to worse new solutions is progressively changed towards zero so that the algorithm will eventually converge to an optimum. With the probability being set and reduced appropriately, the approach is quite likely to find a global optimum.

In practice, for a maximisation problem with the objective function f, the probability p is usually defined as follows:

$$p = \begin{cases} 1 & if \quad \Delta E > 0 \\ e^{\frac{\Delta E}{T}} & if \quad \Delta E \leq 0 \end{cases},$$

where ΔE is the measure of the badness of moving from the current solution x to the successor x', namely $\Delta E = f(x') - f(x)$. The T is the strategy parameter which is called "temperature". Temperature T is usually set high at the start so that the moves that have large badness ΔE are allowed most likely. While the algorithm is running iteratively, T goes down according to the running time or number of iterations, which correspondingly reduces the chance of moving towards worse solutions. Algorithm 5 illustrates the concept of the approach, where the relation between T and iteration number t is represented in a predefined function $schedule(t)$.

3.3.3 Evolutionary Algorithms

An evolutionary algorithm (EA) is a category of meta-heuristics that is inspired by biological evolution, which involves mechanisms such as reproduction, mutation,

Algorithm 6 A Generic Scheme of an Evolutionary Algorithm [48]

```
procedure EVOLUTIONARY ALGORITHM
    INITIALISE population with random candidate solutions
    EVALUATE each candidate
    repeat
        SELECT parents
        RECOMBINE pairs of parents
        MUTATE the resulting offspring
        EVALUATE new candidates
        SELECT individuals for the next generation
    until Termination condition is satisfied
end procedure
```

recombination (also termed *crossover* or *mating*), and selection [48]. The underlying idea of EA is the competition among individuals of a population within a given environment having limited resources, where the survivors of such a competition demonstrate their best fitness within the environment. By defining a problem (especially an optimisation problem) as an environment, an EA imitates the natural selection in biological evolution. After a sufficient number of generations, the ultimate survivors are the good candidate solutions to the problem.

In an EA, the environment is usually defined by a *fitness function*, which evolves a particular candidate solution (or individual). In some cases, the fitness function is directly equivalent to the definition of the problem, i.e. the objective function plus the constraints, but they are not necessarily identical. Fitness defines the direction in which the population will evolve, which forms the basis of selection. Given an objective function to be maximised as the fitness, an EA starts with a set of candidate solutions (namely a *population*) that are randomly created. Each individual of the population is then measured by their degree of fitness. The relatively greater ones usually have more chance of being selected to seed the next generation (namely the offspring), which is the mechanism named reproduction. Mutation and recombination are applied as two operations in this period. The former is unary, which forms a new candidate by modifying a slight part of a selected individual. The latter is usually binary, combining the information from two selected candidates (namely parents) to produce one or more new candidates (namely children). This process is iterated until one or more candidates with sufficient quality is found unless another terminal condition is reached. Algorithm 6 reveals the general scheme of this process.

In general, the following six components must be specified in order to form a particular EA:

- Fitness function,
- Representation,
- Population,
- Parent selection mechanism,

- Variation operators: recombination and mutation,
- Survivor selection mechanism.

Among the above, the representation maps candidate solutions of the original problem (i.e. phenotypes) into the individuals of an EA (i.e. genotypes or chromosome). For instance, a phenotype of the 0-1 knapsack problem (Section 2.3.2) is a packing plan that is a description of the selected items, and the genotype can be a binary string where each bit represents whether or not (1 or 0) the corresponding item is selected. According to different types of representation, EAs are commonly categorised into various forms, as follows:

- **Genetic Algorithm**: the genotypes are strings over a finite alphabet, typically binary or integers;
- **Evolution Strategies**: the genotypes are real-valued vectors;
- **Genetic Programming**: the genotypes are trees;
- **Evolutionary Programming**: the genotypes are finite state machines.

A population, in almost all the cases, is just a set of genotypes. This set usually has a fixed size, where the worst individuals will be eliminated in the survivor selection if the size limit is reached. In addition, some special mechanisms or structures may be incorporated in order to improve the diversity within a population.

The variation operators, namely the binary recombination and the unary mutation, are used to produce the offspring. In the terminology of EA, a $(\mu + \lambda)$EA denotes an evolutionary algorithm with a population size of μ and an offspring size of λ. The parent and survivor selections are two mechanisms that are used to push the quality improvements of the population. The former is used to choose parents to seed the next generations and is typically stochastic. The latter is for the elimination of the worst individuals and often deterministic.

In practice, the decisions associated with the components of EA are often made according to their investigational purpose and the specification or character of the problem. For example the (1+1)EA, which is the simplest EA with a population size

	TSP	KP
Representation	Permutation	Binary strings of length m
Recombination	Edge recombination	One point crossover
Recombination probability	100%	70%
Mutation	Inverse	Bit Flip
Mutation probability	80%	$1/m$
Parent selection	Best 2 out of random 5	Best 2 out of random 3
Survival selection	Replace worst	Replace worst
Population size	100	300
Termination condition	20, 000 generations	No improvement in last 100 generations

TABLE 3.1: Examples of EA components for the travelling salesman problem and knapsack problem

Algorithm 7 A Simple Memetic Algorithm [48]

```
procedure MEMETIC ALGORITHM
    INITIALISE population
    EVALUATE each candidate
    repeat
        SELECT parents
        RECOMBINE to produce offspring
        MUTATE the resulting offspring
        EVALUATE the offspring
        IMPROVE offspring via local search
        SELECT individuals for the next generation
    until Termination condition is satisfied
end procedure
```

of 1 and an offspring size of 1, is usually used to gain some insights into a particular design without influence of too much diversification. In contrast, an EA with well-tuned μ and λ normally aims for good performance. Table 3.1 demonstrates examples of the decisions used for the travelling salesman problem (Section 2.3.1) and knapsack problem.

3.3.4 Memetic Algorithm

Memetic Algorithm (MA) is a type of hybrid approach that couples an evolutionary algorithm (EA) with an individual learning procedure capable of performing local refinements, such as a local search. It is motived by the view that, whilst EAs are good at rapidly identifying good areas of the search space (exploration), they are less competent at fine-turning solutions (exploitation), partially due to the stochastic nature of the variation operators [48].

Inspired by Dawkins' notion of a meme, MA is introduced by Moscato et al. [126] in his technical report where memes can be viewed as agents that can transform a candidate solution that is of direct interest but involves certain utility acquired traits. Such acquired traits are developed by the additional learning or improvement phase incorporated into the evolutionary cycle. Despite there being debate in biology as to whether or not some acquired characteristics are actually inheritable [93], increased interest in this type of hybrid algorithms has been justified by both theoretical and empirical results [48]. Nowadays, MA covers a wide range of techniques where an evolutionary-based search is augmented either by the addition of one or more phases of local search or by the use of problem-specific information. Examples include hybrid GAs, Baldwinian EAs, Lamarckian EAs, genetic local search algorithms, etc. Algorithm 7 represents a simple memetic algorithm.

3.3.5 Swarm Intelligence

Swarm intelligence is a collection of decentralised self-organised systems that typically consist of a population of simple agents interacting locally with one another and within their environment. Similar to the EA, the inspiration for most swarm

intelligence comes from biological systems, such as an ant colony, bee colony, bird flock, etc. Even though that each individual agent follows very simple rules and there is no centralised control structure dictating individual behaviour, the interactions between such agents lead to the emergence of certain intelligential global behaviour.

One famous swarm intelligence system is the ant colony optimisation algorithm (ACO) [44], which is commonly used for solving computational problems that can be reduced to finding good paths through graphs, such as the travelling salesman problem. ACO mimics the behaviour of ants in a colony finding a target (such as food). In nature, an ant lays down a pheromone trail when wandering around. If another ant finds such a path, it is likely to follow the trail instead of wandering randomly, which may reinforce the trail. However, the pheromone trail may evaporate over time. This phenomenon makes ants stay on the shorter path from their colony to the target, as the shorter path is, the least time it takes for an ant to travel down and back again, hence the evaporating pheromones are repaired more frequently. Eventually the pheromone density becomes higher on shorter paths than longer ones. Moreover, when one ant finds a short path from the colony to a target, other ants are more likely to follow that path, and positive feedback eventually leads to all the ants following the single path. In ACO, artificial ants are created in a graph, acting similarly to their counterparts in nature. Therefore, the shortest path can be found after a period of imitation. Other swarm intelligence systems, such as stochastic diffusion search [19], particle swarm optimisation [85] etc., imitate different swarm systems for different types of problems.

3.4 Multi-objective Optimisation

As introduced in Section 2.1.2, a nontrivial multi-objective optimisation problem (MOP) has a set of incomparable solutions that are not dominated by each others, i.e. a Pareto optimal front. Traditional single-solution approaches to solve the MOPs, therefore, are not as straightforward as for conventional single objective optimisation problems. Indeed, a decision has to be made to choose one current incumbent solution among the conflicting ones. Such a choice is commonly made according to certain higher-level information, such as human preferences. According to the way of integrating into the higher-level information, traditional approaches of multi-objective optimisation are usually divided into four categories: no preference, priori, posteriori and interactive methods [82]. As the name suggests, the no preference method does not include extra information for making decisions; the priori and posteriori utilise extra knowledge before or after optimisation respectively; and the interactive method integrates the preference tightly into the process of optimisation.

On the other hand, the population-based approaches, such as evolutionary algorithms etc., provide alternatives that naturally support the optimisation of an entire set of solutions to approximate the Pareto optimal front. This is why they have attracted significant attention from the research community recently [26]. Correspondingly, an evolutionary algorithm for multi-objective optimisation is called a multi-objective evolutionary algorithm (MOEA).

In general, most MOEAs are designed with an assumption to deal with two major inherent issues: the approximation to the Pareto optimal front and a good spread

among the solutions [41]. Many MOEAs achieve the former via assigning fitness to individuals, and the latter is kept by preserving diversity among solutions of the same non-dominated front. On the other hand, there are approaches that focus on the integration of user preferences into MOEA. In this section we briefly introduce three popular MOEAs: NSGA-II [42], SPEA-2 [183] and IBEA [182], each of which demonstrates design of the fitness and diversity preservation.

3.4.1 Non-dominated Sorting Genetic Algorithm II

Non-dominated Sorting Genetic Algorithm II (NSGA-II) [42] is an elitism approach, where the best individuals are preserved during the optimisation process. Some empirical results [181] suggest that the design helps to achieve better convergence in MOEAs. Moreover, NSGA-II proposes a fitness assignment based on sorting the non-dominated solutions, named the fast non-dominated sorting, which is an improvement on naive non-dominated sorting. In terms of diversity preservation, it introduces a crowdedness-comparison operator, which estimates the density of individuals via crowding distance, and guides the selection process toward a uniformly spread-out Pareto optimal front.

Naively, given a population of size N, the sorting of non-dominated solutions works as follows:

(a) compare each solution with every others in the population to find if it is dominated;

(b) label all the non-dominated solutions as being in the first non-dominated front;

(c) repeat the comparison of (a) whilst not taking the solutions of the first non-dominated front into consideration, and label the non-dominated solutions as being in the second non-dominated front;

(d) do the rest in the same manner until all the solutions are labeled as different levels of non-dominated fronts.

In the worst case, when there are N fronts and each front contains only one solution, the overall complexity is $\mathcal{O}(MN^3)$. Deb et al. [42] propose an improved version of the sorting, whereby they store the solutions that each solution p dominates, and the number of solutions which dominate the solution p, to speed up the procedure to become $\mathcal{O}(MN^2)$.

In order to maintain a good spread in the obtained set of solutions, the crowding distance is introduced to estimate the density of the solutions surrounding a particular solution in the population, which is the average distance of two points on either side of this point for every objective. Based on the crowding distance, a crowdedness-comparison operator then guides the selection process towards choosing the solutions that are located in a less crowded region when the candidates are at the same level of non-dominance.

Algorithm 8 Strength Pareto Evolutionary Algorithm 2 [183]

Input N: population size; $\overline{N}$: archive size; T: maximum number of generations
Output A: non-dominated set

1: *Initialisation*: Generate an initial population P_0 and create the empty archive (external set) $\overline{P_0} = \emptyset$. Set $t = 0$.
2: *Fitness assignment*: Calculate fitness values of individuals in P_t and $\overline{P_t}$.
3: *Environmental selection*: Copy all non-dominated individuals in P_t and $\overline{P_t}$ to $\overline{P}_{t+1}$. If size of $\overline{P}_{t+1}$ exceeds $\overline{N}$ then reduce $\overline{P}_{t+1}$ by means of the truncation operator, otherwise if size of $\overline{P}_{t+1}$ is less than $\overline{N}$ then fill $\overline{P}_{t+1}$ with dominated individuals in P_t and $\overline{P_t}$.
4: *Termination*: If $t \geq T$ or another stopping criterion is satisfied then set A to the set of decision vectors represented by the non-dominated individuals in $\overline{P}_{t+1}$. Stop.
5: *Mating selection*: Perform binary tournament selection with replacement on $\overline{P}_{t+1}$ in order to fill the mating pool.
6: *Variation*: Apply recombination and mutation operators to the mating pool and set $\overline{P}_{t+1}$ to the resulting population. Increment generation counter $(t = t + 1)$ and go to Step 2.

3.4.2 Strength Pareto Evolutionary Algorithm 2

Strength Pareto Evolutionary Algorithm 2 (SPEA2) [183] is another elitism approach. Its fitness is based on the strength value $S(i)$, which represents the number of dominated solutions by the individual i. Accordingly, the fitness of an individual j is determined by the sum of the strengths of all the dominators of j. The individuals having a lower fitness value are better in SPEA2.

Similar to the crowding distance of NSGA-II, SPEA2 incorporates the distance to the k-th nearest individuals as the measure of density, where k is commonly set as the square root of the sample size. For each individual i, SPEA2 stores and sorts all the distances from it to another individual. The inverse of the distance to the k-th nearest neighbours is therefore used as the density estimation of i. Algorithm 8 depicts the general procedure of the algorithm.

3.4.3 Indicator-based Evolutionary Algorithm

Despite the popularity of NSGA-II and SPEA2 as well as other MOEAs that focus on addressing both of the assumed issues, Zitzler et al. [182] argue that the assumption is problematic in terms of both definition and practice. Moreover, they believe the popular design of MOEAs does not allow for flexibility with respect to the preference information used. They therefore propose a general indicator-based evolutionary algorithm (IBEA) that can be combined with arbitrary indicators. Such indicators can be adapted to user preferences and the diversity preservation mechanism is not necessarily required.

The authors of IBEA propose a binary quality indicator I to compare the quality of two Pareto set approximations relative to each other. Particularly, if S is the Pareto optimal front and A is the approximation of S, $I(A, S)$ measure the extent to which

Algorithm 9 Basic Indicator-based Evolutionary Algorithm [182]

Input α: population size; $\overline{N}$: maximum number of generations; κ: fitness scaling factor

Output A: Pareto set approximation

1: *Initialisation*: Generate an initial population P of size α; set the generation counter m to 0.
2: *Fitness assignment*: Calculate fitness values of individuals in P, i.e. for all $x_1 \in P$ set $F(x_1) = \Sigma_{x_2 \in P \setminus \{x_1\}} - e^{-I(\{x_2\},\{x_1\})/\kappa}$.
3: *Environmental selection*: Iterate the following three steps until the size of population P does not exceed α:
   1. Choose an individual $x^* \in P$ with the smallest fitness value, i.e., $F(x^*) \leq F(x)$ for all $x \in P$.
   2. Remove x^* from the population.
   3. Update the fitness values of the remaining individuals, i.e., $F(x) = F(x) + e^{-I(\{x^*\},\{x\})/\kappa}$ for all $x \in P$.
4: *Termination*: If $m \geq N$ or another stopping criterion is satisfied then set A to the set of decision vectors represented by the non-dominated individuals in P. Stop.
5: *Mating selection*: Perform binary tournament selection with replacement on P in order to fill the temporary mating pool P'.
6: *Variation*: Apply recombination and mutation operators to the mating pool P' and add the resulting offspring to P. Increment generation counter $(m = m + 1)$ and go to Step 2.

A is close to the Pareto optimal. IBEA is built based on the idea of minimising $I(A, S)$ in a MOEA. Algorithm 9 illustrates the general version of this approach, where fitness $F(x_1)$ is a measure for the loss in quality if x_1 is removed from the population.

3.5 Existing Approaches for the TTP

In this section, we provide a brief overview of existing approaches to the travelling thief problem (TTP), many of which have recently been reviewed by Wagner et al. [169].

In the original article about the TTP, Bonyadi et al. [24] provides a brute-force approach to their demonstrative tiny TTP instances, which illustrates the complexity of this problem. Polyakovskiy et al. [139] develop this problem by proposing a library with $9,720$ instances and a simple set of heuristics for solving them. The essential idea of the heuristics is to solve the problem in two phases: the first is to construct a good TSP tour without considering the knapsack part of the TTP problem; the second is to create a packing plan in order to achieve a good TTP objective value for a given TSP tour. It is noticeable that this approach does not take the interdependence of the two sub-problems into consideration.

By contrast, Bonyadi et al. [25] propose a heuristic that takes interdependence into account. The so-called CoSolver approach is inspired by the cooperative co-evolution, which solves the two sub-problems by two different modules, while the communications between them are exchanged and managed. A comparison across several instances shows the superiority of CoSolver over algorithms without consideration of interdependence.

Mei et al. [117] furthermore highlight the necessity of taking interdependence into account. Their mathematical analysis shows that the TTP problem is not additively separable, which results in the view that one cannot expect to achieve competitive results by solving each component in isolation. The authors propose a memetic algorithm with a two-stage local search, which considers the interdependencies in more depth. It outperforms the cooperative coevolution based approach. Mei et al. [116] conduct a more systematic investigation into the item picking heuristic in the two-stage memetic algorithm, which yields a better approach. In [118], they investigate the interdependence of TSP and KP in TTP more theoretically and empirically, which demonstrates that considering the interdependence between sub-problems is important for obtaining high-quality solutions for the overall problem.

Investigating the multiple operators of existing approaches comprehensively, Faulkner et al. [56] propose a number of new operators, such as BitFlip and Pack-Iterative, for optimising the packing plan given a particular tour; and Insertion for iteratively optimising the tour given a particular packing. The heuristics that consist of these operators outperforms the existing approaches.

Strzeźek et al. [160] propose a diversity-aware population selection operator for genetic algorithms, which shows potential for improving the quality of the results for the TTP. The operator enables the possibility of controlling the balance between exploration and exploitation by managing the dispersion of the solutions.

Realising that the knapsack sub-problem has generally been found to have greater impact on the objective function when a near optimal tour is provided, Gupta et al. [69] designed the greedy heuristics that focus on solving the knapsack part, based on the classic greedy approaches for the KP.

Wagner [168] investigates the use of swarm intelligence approaches with the so-called Max–Min Ant System (MMAS, by Stützle et al. [161]). The resulting approaches focus on getting better overall rewards without being constrained by shorter TSP tours. This allows them to outperform the previous best approaches by Faulkner et al. [56] and Mei et al. [117] on relatively small instances with up to 250 cities and 2000 items.

El Yafrani et al. [51] discuss the pros and cons of a local search implementation to solve the TTP. They analyse the issues of implementing local search algorithms to solve a multi-component problem and propose an approach to implement neighbourhood deterministic local search algorithms to solve small and mid-size TTP instances. In their other paper, El Yafrani et al. [50] study and compare different approaches for solving the TTP from a meta-heuristics perspective. A memetic algorithm (MA2B) and one using simulated annealing (CS2SA) are proposed. Their results show that the algorithms were competitive with the approaches by Faulkner et al. [56] and Mei et al. [117] on a range of larger TTP instances. They also investigate the on a 2-OPT steepest ascent hill climbing algorithm and the simulated

annealing meta-heuristic [52]. Their results are very competitive in comparison with the state-of-the-art algorithms.

Martins et al. [113] consider a hyper-heuristic framework which encompasses a heuristic selection approach aiming to find the best combinations of different known heuristics according to a probabilistic distribution model. This so-called Heuristic Selection based on the Estimation of Distribution Algorithm (HSEDA), outperforms the state-of-the-art algorithms in most of the medium-sized TTP instances considered in their paper. On the other hand, El Yafrani et al. [53] study another hyper-heuristic framework of heuristic generation methodologies, which aims to automatically design new heuristics using components of previously known heuristics [157]. Their results show that this genetic programming (GP [94]) based approach is competitive with the state-of-the-art algorithms on small and mid-sized TTP instances.

Lourenço et al. [104] present an evolutionary approach for the TTP, which seeks complete solutions by simultaneously considering the two sub-problems and the existing interdependence between them. Their EA relies on the individuals each having a tour and a packing plan. The variation operators modify both components, and a packing heuristic helps creating good packing plans for each individual.

On the other hand, studies on the multi-objective versions of the TTP are relatively limited. Blank et al. [20] investigate a variant of the TTP, which is a bi-objective version of the TTP, i.e. maximising the total profit and minimising the overall transportation costs. Yafrani et al. [178] create an approach that generates diverse sets of Model I/II solutions, while being competitive with the state-of-the-art single-objective algorithms.

3.6 Conclusion

Chapters 2 and 3 cover the prerequisites and backgrounds of the studies in this thesis. In this chapter, we have introduced the general approaches to solving optimisation problems and listed some examples, which include exact approaches such as dynamic programming, branch and bound method and constraint programming; heuristics such as local search, simulated annealing, evolutionary algorithms and etc,. as well as three popular approaches for multi-objective optimisation. In addition, the approaches proposed in the existing literature for the travelling thief problem have been briefly reviewed. From the next chapter on, we will start to introduce our theoretical and empirical investigations on complex multi-objective optimisation problems.

Chapter 4

Fast and Effective Optimisation of Arrays of Submerged Wave Energy Converters

In this chapter, we elaborate the case study of wave energy converter (WEC) optimisation that is introduced in Section 2.2.1. Here, we try to optimise the overall energy absorption of an array of WECs. In order to achieve this objective, not only are the parameters of each individual WEC to be optimised according to the hydrological frequencies in the ocean area, but we also take into consideration the hydrodynamic interaction between each device, because these interactions have a significant impact on the overall performance of the array [153], optimising each silo individually cannot yield the array's overall optimality. However, due to the interaction of its multiple components in this problem, the computational cost of the evaluation of the model is prohibitively expensive, taking hours or even days for just one evaluation. Iterative optimisation approaches, such as EA, are hardly to be applied in this problem. We therefore creatively simplify this model via an approximation, which makes the iterative optimisation doable.

In the current body of research on WEC arrays and their optimisation, many of the devices under consideration are semi-submerged or floating [57, 13, 29]. In contrast, the CETO WEC is fully submerged beneath the ocean's surface [106], as this increases survivability in high sea states and has almost no visual impact. There is very limited research into fully submerged wave energy converters. In particular, we are not aware of any research into optimising the placement or configuration of arrays of fully submerged wave energy converters. Moreover, a technological alternative to single-tether CETO WECs are three-tether WECs, as shown in Figure 4.1. The capital cost of such devices is higher than that of conventional single-tether heaving buoys due to the increased number of separate power take-off (PTO) systems for each tether. The total cost of the three-tether WEC array can be reduced significantly if the layout allows adjacent devices to share the same mooring points (see Figure 4.2). In this study, we will investigate and compare array layouts both with and without shared mooring points.

In order to evaluate arrays, we use a recently developed frequency domain model for arrays of fully submerged three-tether WECs. This model allows us to investigate different parameters, such as the number of devices, array layout and buoy size. The ideal choice of parameters leads to an optimisation problem: what is the best combination of buoy radii and their locations for different array sizes? This question therefore becomes the decision variables of this problem.

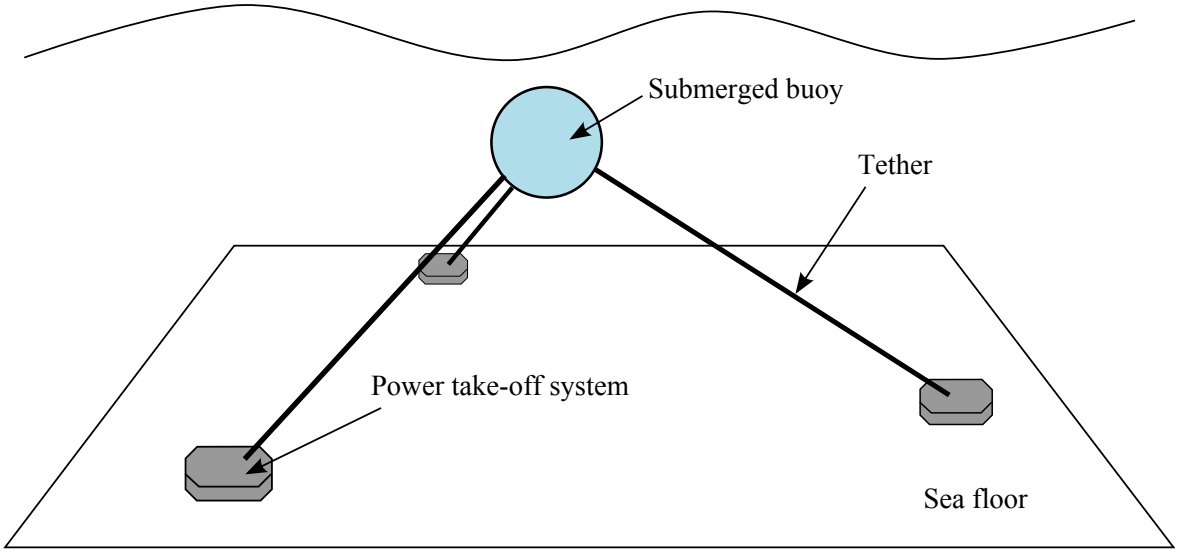


FIGURE 4.1: Schematic representation of a three-tether WEC (adapted from [28]).

This chapter is based on a conference paper published at the Genetic and Evolutionary Computation Conference (GECCO) [175]. The structure of the chapter is as follows: we introduce the foundation model of interacting three-tether buoys in Section 4.1 that we base our investigations on. In Section 4.2, we describe our speed-ups of the original model, as the original model is computationally prohibitively expensive for use in iterative optimisation approaches. Then, we present our experimental results in Section 4.3, and finish with our concluding remarks.

4.1 Model of the Three-Tether WEC Array

4.1.1 System description

The WEC design that we consider is a fully submerged spherical body connected to three tethers that are equally distributed around the buoy hull (Figure 4.1). Each tether is connected to the individual power generator at the sea floor, which allows to extract power from surge and heave motions simultaneously [152].

The arrangement of a three-tether WEC array may be considered in two different ways:

(i) In arrays where all adjacent devices share common anchorage points and/or power take-off system (see Figure 4.2). The main benefit of this layout is a significant reduction in the capital cost of the array due to the smaller number of mooring points as compared to the separately placed WECs. At the same time, the optimal buoy placement in such arrays is fixed and depends only on the ocean depth at the particular sea site and desired submergence depth of the buoy [153];

(ii) In arrays where all devices are placed separately (see Figure 4.1). This layout does not have any constraints on the farm geometry and a buoy placement can be chosen considering various optimisation procedures.

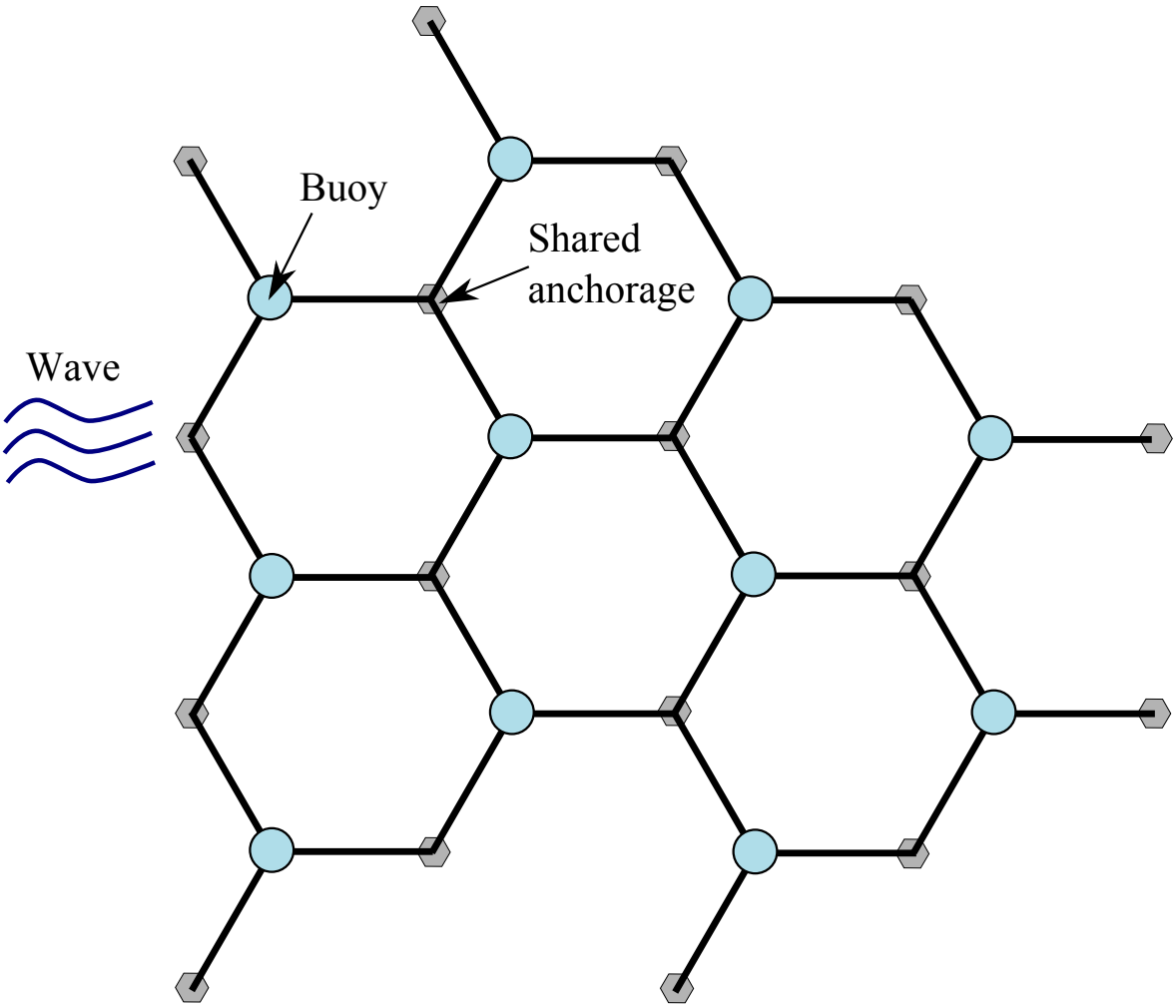


FIGURE 4.2: Top view on the array of WECs with shared mooring points.

4.1.2 System dynamics

In the following, we briefly outline the model of this kind of WECs arrays as it was derived by Sergiienko et al. [153].

The dynamic equation of the WECs array is derived in the frequency domain using linear wave theory, where a fluid is inviscid, irrotational and incompressible [55]. This model considers three dominant forces that act on the WECs:

(i) excitation force includes incident and diffracted wave forces when all bodies are assumed to be fixed;

(ii) radiation force acts on the oscillating body due to its own motion in the absence of incident waves;

(iii) control, or power take-off (PTO) force, that exerts on the WEC from machinery through tethers.

The key point in the array performance is the hydrodynamic interaction between buoys that can be constructive or destructive depending on the array size and geometry.

A spherical body is excited by ocean waves in surge, sway and heave only [158, 102]. However, a geometrical arrangement of a WEC with three tethers induces small angular motions of the body that do not contribute to the power absorption. Therefore, only translational motion of each body is included in the dynamic equation of the system.

Assuming that the total number of devices in the array is N and p is the body number, then the dynamics of the p-th WEC in time domain is described as:

$$\mathbf{M}_p \ddot{\mathbf{x}}_p(t) = \mathbf{F}_{exc,p}(t) + \mathbf{F}_{rad,p}(t) + \mathbf{F}_{pto,p}(t), \tag{4.1}$$

where $\mathbf{M}_p$ is a mass matrix of the p-th buoy, which is presented by the product of the buoy's mass m_p and an identity matrix in the size of 3, i.e. $\mathbf{M}_p = m_p I_3$, in order to calculate the forces on the 3 dimensions respectively; $\ddot{\mathbf{x}}_p(t)$ is a body acceleration vector in surge, sway and heave, $\mathbf{F}_{exc,p}(t)$, $\mathbf{F}_{rad,p}(t)$, $\mathbf{F}_{pto,p}(t)$ are excitation, radiation and PTO force vectors respectively. The PTO system is modelled as a linear spring and damper for each mooring line with two control parameters, such as stiffness K_{pto} and damping coefficient B_{pto}.

In case of multiple bodies, where $p = 1 \dots N$, Equation (4.1) can be extended to include all WECs and expressed in frequency domain:

$$\left(\left(\mathbf{M}_\Sigma + \mathbf{A}_\Sigma(\omega)\right) j\omega + \mathbf{B}_\Sigma(\omega) - \frac{\mathbf{K}_{pto,\Sigma}}{\omega} j + \mathbf{B}_{pto,\Sigma} \right) \hat{\dot{\mathbf{x}}}_\Sigma = \hat{\mathbf{F}}_{exc,\Sigma}, \tag{4.2}$$

where subscript Σ indicates a generalised vector/matrix for the array of N bodies, $\mathbf{A}_\Sigma(\omega)$ and $\mathbf{B}_\Sigma(\omega)$ are radiation added mass and damping coefficient matrices that include hydrodynamic interaction between buoys, $\mathbf{K}_{pto,\Sigma}$, $\mathbf{B}_{pto,\Sigma}$ are the stiffness and damping block-matrices of the PTO system.

As a result, knowing such parameters as coordinates and dimensions of each buoy within the array layout and ocean depth at a particular sea site, the hydrodynamic coefficients of the interactions can be calculated using specialised packages (e.g. WAMIT [97]) or by using analytical models. Moreover, specifying control parameters for each PTO system, motion amplitudes and velocities of all buoys can be determined using equation (4.2).

4.1.3 Performance index

The total power absorbed by the array of WECs can be calculated as:

$$P_\Sigma = \frac{1}{4}(\hat{\mathbf{F}}^*_{exc,\Sigma}\hat{\dot{\mathbf{x}}}_\Sigma + \hat{\dot{\mathbf{x}}}^*_\Sigma \hat{\mathbf{F}}_{exc,\Sigma}) - \frac{1}{2}\hat{\dot{\mathbf{x}}}^*_\Sigma \mathbf{B}\hat{\dot{\mathbf{x}}}_\Sigma, \tag{4.3}$$

where * denotes the conjugate transpose.

The performance of an array of N WECs is usually summarised using the so-called q-factor:

$$q = \frac{P_\Sigma}{N \cdot P_0}, \tag{4.4}$$

where P_0 is the power absorption of a single device in isolation. The q-factor is the ratio of the power absorption of an array of WECs compared to the power absorption of those same converters in isolation. A q-factor greater than one indicates the

presence of constructive interference in the array, as the array of devices is producing more energy than the devices would individually. Conversely, a q-factor less than one is a sign of destructive interference, which may be detrimental to the performance of the array.

Lastly, for the fair analysis of layouts that involve WECs of different sizes, we choose the *relative capture width* (RCW) to be a non-dimensional index of power absorption:

$$RCW = \frac{P_\Sigma}{P_w \left(2 \sum_{p=1}^{N} a_p \right)}, \tag{4.5}$$

where P_w is the incident wave-energy transport per unit frontage, a_p is a radius of the p-th body. RCW shows the fraction of power extracted from the wave per unit length of the device. RCW from Equation (4.5) is frequency dependent, therefore, for the particular sea site location, the RCW should be weighted according to the sea state probability data. Thus,

$$\overline{RCW} = \frac{\sum_i n_i \cdot RCW(\omega_i)}{\sum_i n_i}, \tag{4.6}$$

where n_i is an occurrence probability of waves at particular frequency.

4.1.4 Model specification

In Table 4.1 we provide the dimensions of the WECs used in the remainder of this article. We choose constant power take-off coefficients to give optimal power for the regular wave of 1m amplitude and 9-second period. The mass of each buoy is equal to 0.85 times the mass of the displaced water. Ocean depth is chosen to be 50m and all WECs are submerged 6m to centre of buoy.

TABLE 4.1: Specification of WECs used in array optimisation.

Buoy radius a, m	5	4	3.2	2.5	2
PTO spring coefficient K_{pto}, kN/m	387	185	92	43	22
PTO damping coefficient B_{pto}, kN/(m/sec)	161	76	38	18	8.9

We calculate the hydrodynamic parameters of the WEC array (excitation force, added mass and damping coefficients) based on the algorithm presented by Wu [172]. The results of various array layouts and buoy sizes have been validated against WAMIT [97], which is a computer program for computing wave loads and motions of offshore structures in waves.

4.2 Array Optimisation

In this section we present our approaches used to speed-up the simulations of the WEC arrays. The techniques include approximations and caching. For an array of 50 WECs, the eventual speed-up is 350-fold, i.e., from approximately 2100 minutes down to six minutes.

4.2.1 Model Approximation

The model approximation $\mathcal{M}'$ is a substitute of the three-tether model $\mathcal{M}$ with significantly reduced computational cost and acceptable error of accuracy. In terms of accuracy, we create a function p to compare the two models only based on the agreement of their trends. In other words, if the benefit is increasing/decreasing in $\mathcal{M}$ when changing from layout l_1 to layout l_2, we compare whether the same trend takes place in the approximation $\mathcal{M}'$. Function p is defined as:

$$p(f(x), x_1, x_2) = \begin{cases} \frac{f(x_1)-f(x_2)}{|f(x_1)-f(x_2)|} & f(x_1) \neq f(x_2) \\ 0 & f(x_1) = f(x_2) \end{cases}$$

Based on the function, a standard binary test is introduced according to the rule that $p(\mathcal{M}', l_1, l_2) = p(\mathcal{M}, l_1, l_2)$ means *positive* and the contrary means *negative*. This way we can compute the accuracy with regard to True Positive, True Negative, False Positive and False Negative.

In order to reduce the computational cost, we consider to reduce the sampling of frequencies. The original variant of the three-tether model utilises 50 sample frequencies to simulate the probability of wave frequencies in reality. In Figure 4.3, the blue histogram shows the records of different wave frequencies with their probabilities taking place in a sea area close to Sydney [74], and in red we illustrate the 50 evenly chosen frequencies. Each point represents a certain small range of wave frequency and its probability is the sum of the probability of this range. Therefore the total probability of 50 frequencies still sums up to 1, so that the approximate power absorbed by WECs can be calculated by using this simplified version. However, the computation of total power is still costly. The calculation of an array with 50 WECs takes around 35 hours on one core of an Intel Core i5-4250U processor. Since the computation time in linear in the number of considered frequencies, a natural way to the reduce computational cost is to approximate the accurate model with fewer sample frequencies. Our goal is to reduce computation time while keeping the accuracy above 80%.

To achieve this, we create the model approximations with the numbers of sample frequencies to be 10, 5, 4, 3, 2 and 1. Each sample frequency represents a range of actually occurring wave frequencies, and for each approximation we distribute them equally over the spectrum. For the single frequency, however, we select the most likely occurring frequency: 0.7 rad/s. Figure 4.4 illustrates all the probabilities of frequencies used in the six approximation models compared with the probabilities in the original three-tether model.

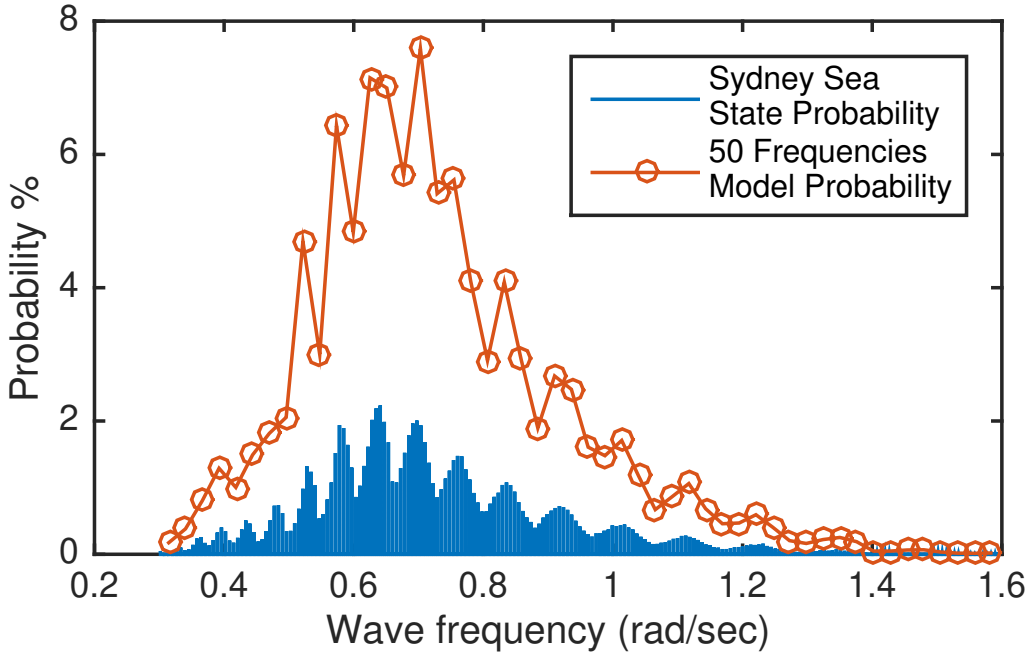


FIGURE 4.3: Sydney sea state: historic distribution and 50 samples

We investigate the six approximations in two specific scenarios: 1) arbitrary layouts and 2) evolving layouts. Both of the scenarios are typical in optimisation, especially for evolutionary algorithms. We study layouts with 50 WECs in a one square kilometre rectangular area with a safety constraint that the minimal distance between each pair of WECs must be 50 meters.

Arbitrary Layouts

In this scenario, we randomly generate 100 valid layouts and divide them into five groups. For each group, we calculate the accuracy between the three-tether model and each of six approximations. Then we plot the averages and standard deviations of the groups of data in Figure 4.5. As we can see, the two- and three-frequency approximations are the least accurate ones. The fastest model that considers only the prevailing frequency is comparable in accuracy with the one that uses fives frequencies, however, the latter takes fives times as long to compute.

Evolving Layouts

In this scenario we use a simple evolutionary algorithm called (1+1)-EA to study the optimisation using the approximating three-tether model. This algorithm is a hill-climber where new solutions are created based on the best-so-far encountered. If the new solution provides a higher score, then it replaces the best-so-far, otherwise the new solution is discarded; this is repeated until the total time budget is used up. The reason of using the simple approach is due to two considerations: 1) achievability, dispite our efforts on speeding up the computation and simplifying the model, the computational cost of evalutating the model is still relatively expensive, which does not allow us to experiment comprehansive approaches; 2) understandability,

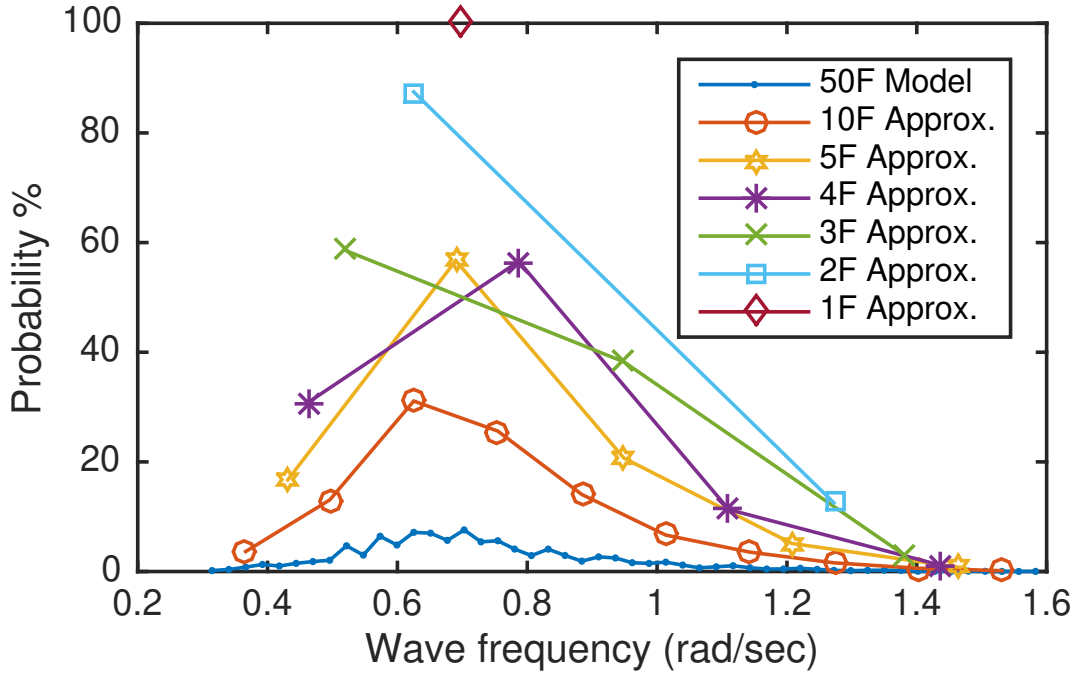


FIGURE 4.4: Probabilities of frequencies in six approximation models and the original three-tether model

simple (1+1)-EA provides clear picture about how itrative optimisation works in our approximation.

We run 400 generations of the algorithm with a simple mutation which randomly chooses and moves only one WEC in a layout. This optimisation results in an increase of the power output by around 5% (as shown in Figure 4.6), and it also generates 401 layouts including an initial random layout. We then calculate the accuracy between the original three-tether model and each of six approximations based on the layouts by using the same approach as for arbitrary layouts. The results are again shown in Figure 4.7.

The results of both scenarios largely agree. The 10 and 5 frequencies approximations provide the best accuracy and precision in both scenarios. However, we do not choose them due to their relatively higher cost compared with the single frequency approximation. The single frequency (i.e., the prevailing frequency) approximation provides acceptable accuracy and precision with minimal cost, which makes it the ideal trade-off in our case. With incorporating the approximation, the cost of computation is reduced by around 98%, i.e., from around 2,100 minutes to be around 42 minutes for calculating one layout of a 50 WECs array.

4.2.2 Model Speed-Up Through Caching

Another approach that we introduce along with the single frequency approximation in order to reduce the computational cost is 'caching', which is a technique widely used in software engineering for improving performance. In our particular model, the most frequently used calculations in our MATLAB model are `integral`, `factorial`, and `bessel`. The time spent with such calculations is significant, and

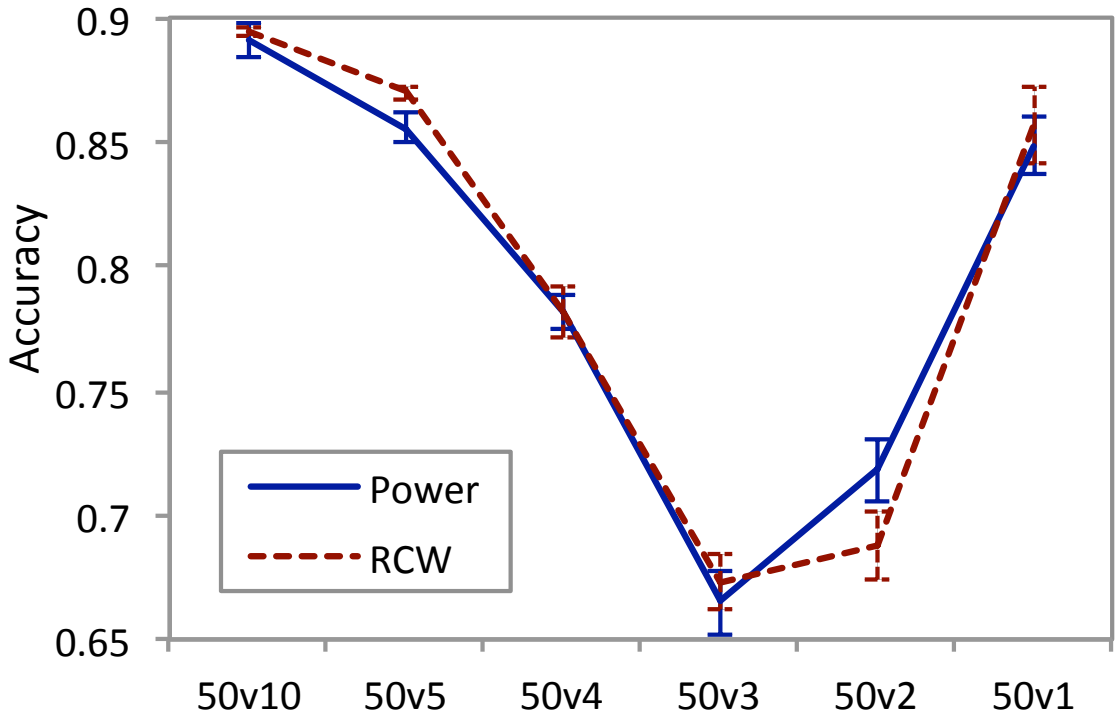


FIGURE 4.5: Accuracy of six approximation models for arbitrary layouts. Shown are the results when 10, 5, 4, 3, 2, and only 1 (of 50) frequencies are used.

a number of them are duplicated during the power computations for a single layout. For instance, in order to calculate the power output of a 50 WECs array, one million calls of `integral` are made, while around 89.5% of them are duplicates. Therefore we cache the results of such calculations into several hash-maps with their parameters hashed to be the corresponding keys. This way, subsequent calls can query the results with their parameters instead of recalculating them. By implementing this technique, the computational cost can be reduced by around 85% without influencing the accuracy. For calculating one layout of a 50 WECs array mentioned in the previous section, the cost is decreased further from about 42 minutes to about 6 minutes.

4.3 Computational Study

In this section we report on our layout investigations of submerged wave energy converters. In the first set of experiments, we consider WECs arranged in a grid-based layout. There, the devices can share mooring points and/or power take-off systems, which results in a significant reduction in capital cost. In the second part, we relax this constraint to investigate layouts where the buoys can be placed arbitrarily, as long as the minimum safety distance is maintained.

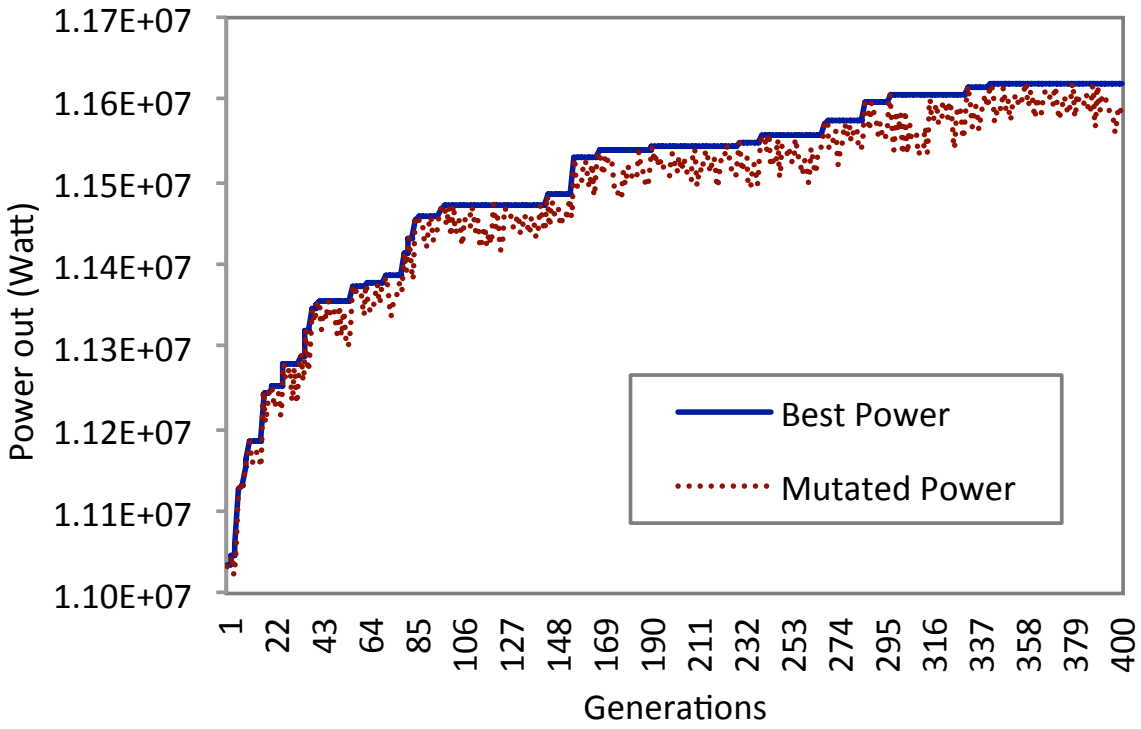


FIGURE 4.6: Optimisation results of (1+1)-EA with a simple mutation

4.3.1 Radii Optimisation

We conduct a range of experiments for optimising the radii of buoys in a staggered array as shown in the introductory Figure 4.2. In this array, the columns of buoys are spaced 93.33m apart and the rows 107.77m, due to technical reasons. Each buoy in the array can have a different radius of either 2m, 2.5m, 3.2m, 4m or 5m. This quantisation is necessary for both optimisation and also in practice, in order to reduce the number of buoy variants. q-factor is primarily used as the optimisation criterion, although some experiments also consider the relative capture width (Equation 4.6).

For small array sizes, including 1x1, 1x2, 2x1 and 2x2, it is feasible to use brute force search (BFS) to explore the entire solution space and find the optimal solution. For example, the largest of these small arrays is the 2x2 configuration, which has 625 possible solutions and takes 10 hours to evaluate them all. The best 2x2 configuration has a q-factor of 0.9990 (with a corresponding $\overline{RCW}$ value of 0.6453), which is a layout comprising of two 2m buoys and two 5m buoys. Interestingly, the best-performing 2x2 layout in terms of $\overline{RCW}$ achieved a significantly higher value of 0.7988 (a layout with four 5m buoys), while the q-factor value decreased slightly to 0.9658, see Figure 4.8. However, this is actually not surprising since the q-factor and $\overline{RCW}$ are two different measures: while $\overline{RCW}$ refers to the maximum power, the q-factor shows to the maximum efficiency of the array in comparison to individual devices. In the model, buoys of different sizes are submerged to the same depth due to constraints of the staggered layout, which affects the efficiency of smaller buoys. Thus, 5m devices submerged to 6 m are more productive in terms of power than 2m buoys submerged to the same depth. As a result, the optimisation using $\overline{RCW}$ ends up with larger WECs. On the other hand, for the q-factor optimisation it is more important to have a constructive hydrodynamic interaction between buoys in

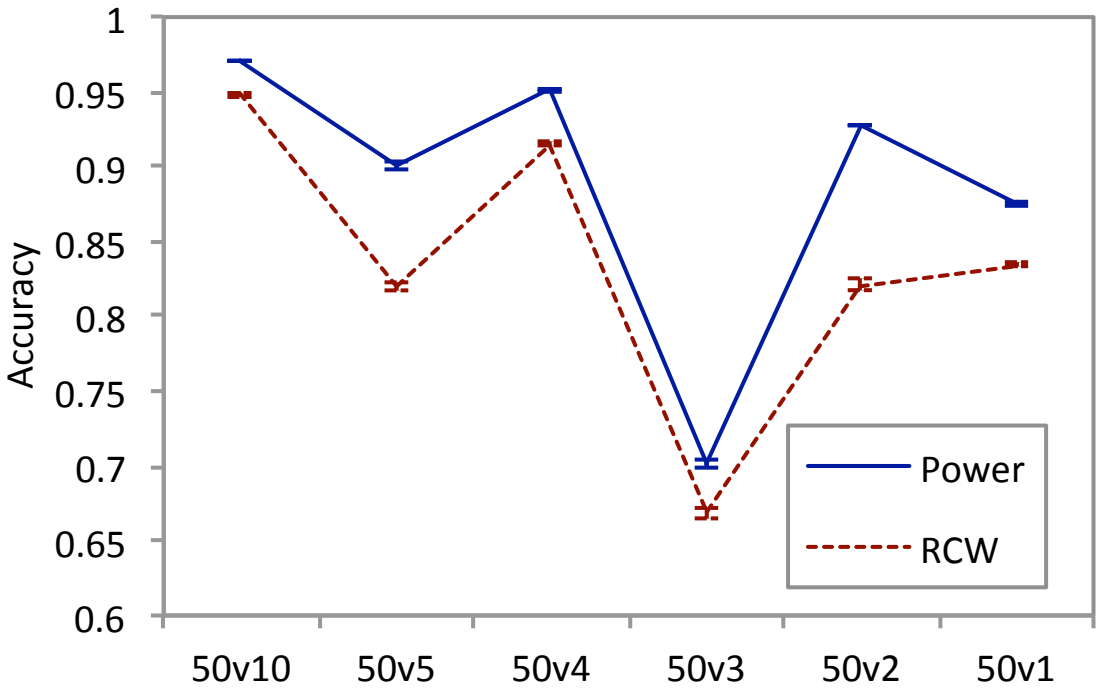


FIGURE 4.7: Accuracy of six approximation models for evolving layouts. Shown are the results when 10, 5, 4, 3, 2, and only 1 (of 50) frequencies are used.

the array. Taking into account that in a staggered layout distances between devices are around 100m, values of q-factor are much higher for smaller buoys as at such distances interaction is reduced to a minimum.

The 3x3 array configuration has almost 2 million solutions, meaning that a brute force search is no longer feasible due to the simulation times needed. Yet for smaller arrays sizes, the optimal configuration is found to only consist of buoys with a radius of either 2m or 5m. Using this insight, we are able to conduct a partial BFS of the 3x3 array by examining only those solutions containing 2m and 5m buoys. This partial BFS takes approximately 2 days to complete, but the result is a solution with a q-factor of 0.9956 (see Figure 4.9), which is comparable to the 2x2 optimal configuration, even though the search was not completely exhaustive in this case.

Since this 3x3 solution found by the partial BFS is not necessarily optimal, we tried using several variants of randomised local search. This did not yield a better 3x3 configuration.An exhaustive evaluation of the local neighbourhood further revealed that this all 2m buoy solution was indeed a local optimum for single changes in the buoy diameters.

We also briefly consider the 4x4 and 5x5 configurations. As BFS has proved to be inefficient, we simply generate all 2m buoy solutions for 4x4 and 5x5, and all of them proved to have q-factors of approximately 0.99. Although these are unlikely to be optimal, the relatively high q-factors show that all 2m buoy solutions may provide configurations with relatively high q-factors for even larger arrays. A similar local neighbourhood check for these 4x4 and 5x5 solutions shows that they are indeed local optima for performing changes to single buoys. This proves that a q-factor is not suitable for the buoy size optimisation at the fixed layout and another performance

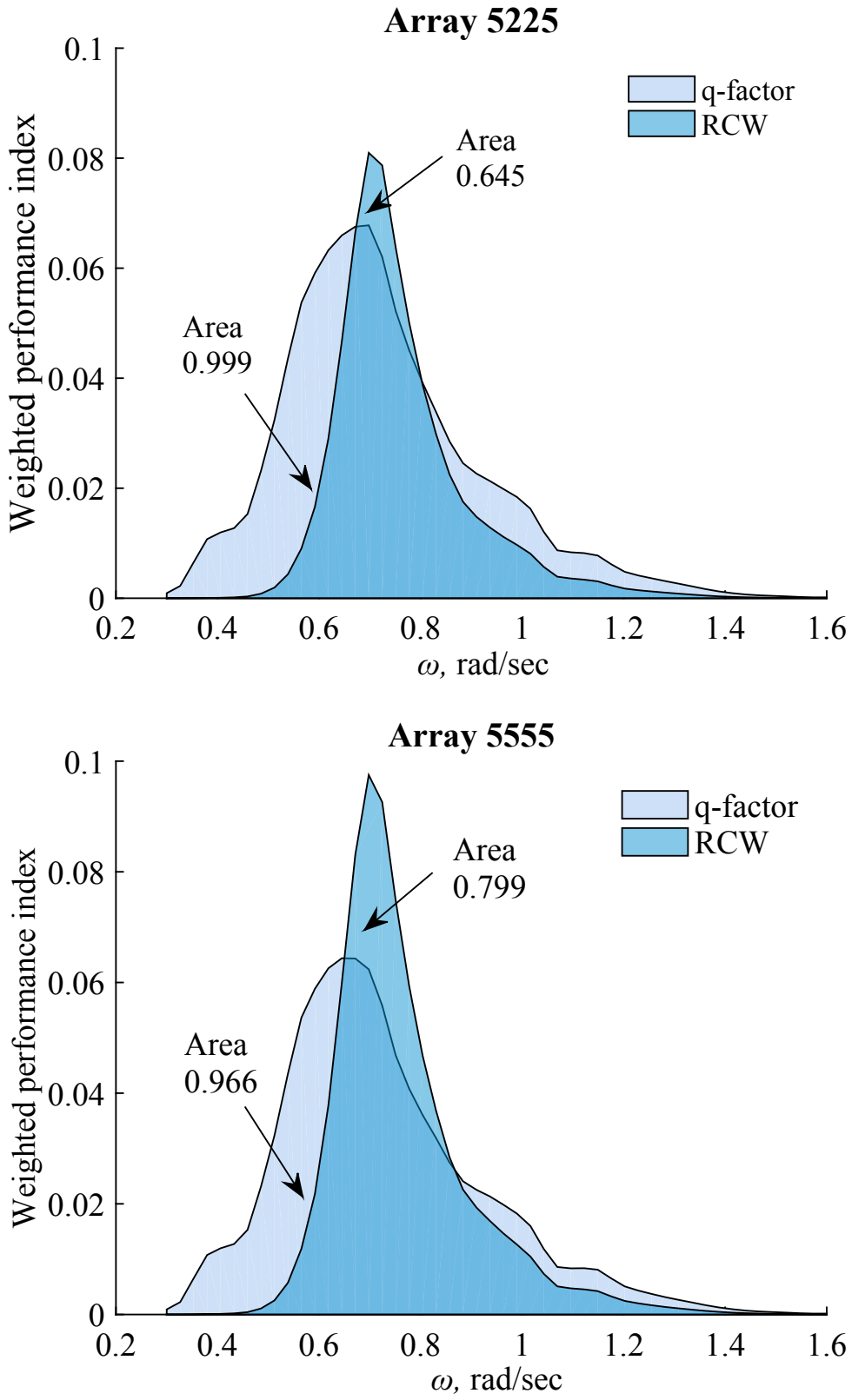


FIGURE 4.8: Comparison of the best performing layouts 5225 and 5555. Shown is the performance for different wave frequencies. The area under the curve corresponds to the expected performance.

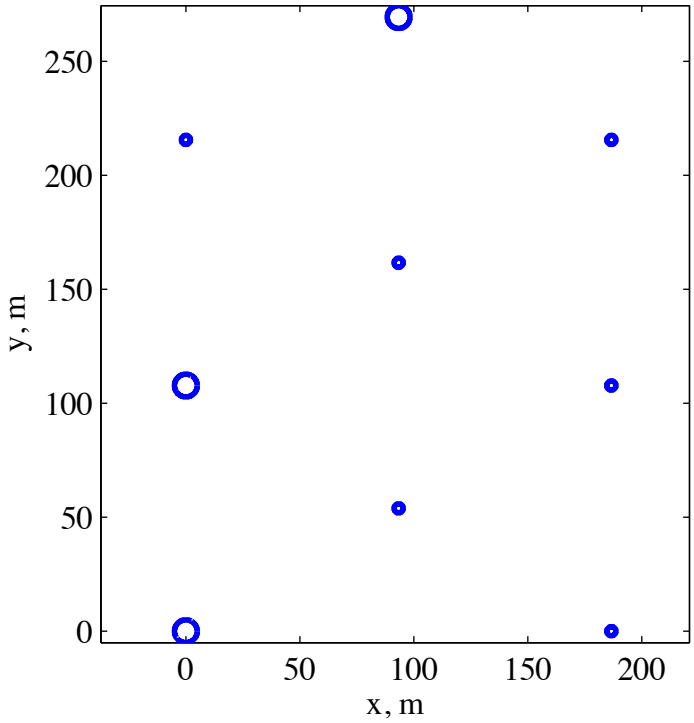


FIGURE 4.9: Best solution found for the 3x3 staggered array. The direction of wave propagation is from left to right. All buoys have diameters of 2m or 5m. q-factor value = 0.9956. $\overline{RCW}$ value = 0.5303. The large 5m buoys make the most of the incoming waves, and the small 2m buoys are most efficient when placed behind the 5m buoys.

index should be developed for such a task.

4.3.2 Placement Optimisation

In the following experiments we no longer enforce the grid-like layout from before. We employ two different algorithms to optimise the layouts. The first one is the (1+1)-EA (as used in Section 4.2), which randomly chooses and moves only one WEC in a layout to a new feasible location. The second algorithm the Covariance Matrix Adaptation based Evolutionary Strategy (CMA-ES) [71]. CMA-ES self-adapts the covariance matrix of a multivariate normal distribution. This normal distribution is then used to sample from the multidimensional search space where each variate is a search variable. The covariance matrix allows the algorithm to respect the correlations between the variables making it a powerful (and popular) heuristic search algorithm.

Initially, both algorithms place the N buoys randomly in the provided area. In preliminary experiments we found that the regular grid initialisation with maximal distances in the rows and columns to perform similar to the random one. While the grid minimises the interactions by maximising the intra-buoy distance, interestingly the positive interferences appear to outweigh what would intuitively be considered a disadvantage.

Both algorithms take care of the constraints in the following ways. When a layout has buoys which violate the proximity constraint or if a buoy is located outside the allowed area, we resample a new solution in (1+1)-EA and CMA-ES, before invoking the time-consuming simulations. For boundary constraints, CMA-ES rounds the coordinates to the nearest boundary value.

The CMA-ES configuration we use here is as follows. We use a population size of two, which is used to generate two new solutions. We run this (2+2)-CMA-ES for 200 generations, and with an initial standard deviation for each decision variable of 20, based on preliminary experiments. The second algorithm, (1+1)-EA, we run with the same total evaluation budget of 400 evaluations.

As we now focus on larger arrays, we use the approximate model from Section 4.2, where only a single frequency is considered. Under the provided conditions, a single isolated 5m buoy has a power output of 5.547e+5 Watts.

Figure 4.10 shows the results from our optimisation of both the simple (1+1)-EA and the CMA-ES. In both cases, the former produces layouts with significantly higher outputs. In the study of 25 buoys, the (1+1)-EA yields the median power of $1.240\text{e}{+7}$ Watts in between the minimum $1.222\text{e}{+7}$ Watts and the maximum $1.249\text{e}{+7}$ Watts, which is much better than the counterpart $1.206\text{e}{+7}$ Watts of CMA-ES within the range $[1.195\text{e}{+7}, 1.231\text{e}{+7}]$ Watts. The same happens in the study of 50 buoys: the (1+1)-EA has the better median power output at $2.220\text{e}{+7}$ Watts in between $2.194\text{e}{+7}$ and $2.237\text{e}{+7}$ than the median $2.151\text{e}{+7}$ Watts within $[2.128\text{e}{+7}, 2.186\text{e}{+7}]$ of the CMA-ES. Both ranges of the results of (1+1)-EA are also narrower than the ranges from CMA-ES, which we take as an indication that (1+1)-EA converges better than CMA-ES, given the evaluation limit provided. Interestingly, this simple algorithm outperforms CMA-ES, even though the later can adapt itself to the problem. It appears that the 200 generations given to CMA-ES are not enough time. To a limited degree this is supported by our observation that CMA-ES begins to converge at the end of the computation budget provided. By then, the average standard deviation decreases to values of about 4 to 8, which means that large changes to the layouts become increasingly unlikely.

As (1+1)-EA is not able to fine-tune a solution, we take a solution found for 25 buoys and give it to CMA-ES for fine-tuning, with $\sigma = 1.0$ for 200 generations. The resulting layout is shown in Figure 4.11 and its power output increased by 1.1%. This means that while CMA-ES experiences difficulties in creating good layouts from scratch, it can still be used to tune existing solutions.

In this layout, it is not very surprising that the buoys facing the incoming waves have the highest power output. Further into the farm, the output decreases quickly, because the interactions become increasingly important with increasing number of columns [7]. This is turn shows the fidelity of our optimisation results. The optimisation considers this indirectly, as the density of the buoys on the left hand side of the final layout is significantly higher than the density of buoys in the right hand side. Interestingly, constructive interferences result at times in individual WECs having an above-average output (greater than $5.547\text{e}{+5}$ Watts) at certain locations, e.g. the buoy located at $(360, 680)$.

Finally, we briefly demonstrate the applicability of our approach to a very large array. In the single run that we perform (1+1)-EA (again using 400 generations) increases the q-factor significantly by 10.4% over the initial layout. Note that the

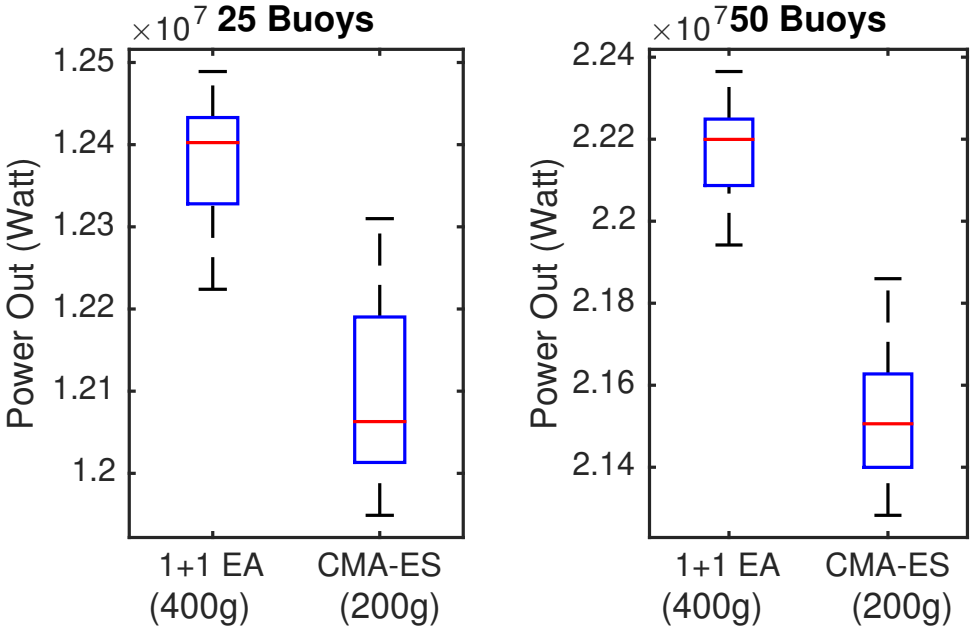


FIGURE 4.10: Optimisation results from our 25 and 50 buoys study. Shown are the results of 20 independent runs. For 25 buoys, the initial average q-factor is 0.8123, and the final q-Factors for (1+1)-EA and CMA-ES are 0.8930 and 0.8723. For 50 buoys, the initial average is 0.7267, and the final ones are 0.7995 for (1+1)-EA and 0.7760 for CMA-ES.

optimisation with the original model would have taken about 2750 days. The actual optimisation using our speed-ups presented in Section 4.2 took only 8.3 days, which corresponds to a speed-up by a factor of 330.

We show the final layout in Figure 4.12. Just as before, the buoys facing the incoming waves have the highest power output. Further into the farm, the output decreases quickly, however, at times positive interferences result in individual turbines having an above-average output.

4.4 Conclusions

This study provides first insights into the layout problem of submerged wave energy converters. It is also the first time multiple three-tether buoys have ever been investigated. Among other key pieces of new knowledge, we have made two high-level observations that enable far more effective design of such arrays. First, we have discovered a potential design flaw, i.e., buoys of different diameters should not be submerged at the same depth, but the top surface of all buoys should be same distance to the sea surface. Second, we have learned that positive interference can result in higher than normal power outputs for individual buoys. This is surprising, since such effects are hardly ever heard of. For example in wind energy related research, wake effects and turbulences with their negative effects are well-known, but positive effects are not. In the chapter on potential future work on the analysis of a WECs array, we suggest that the interaction model might need to be refined to allow for varying submergence depths.

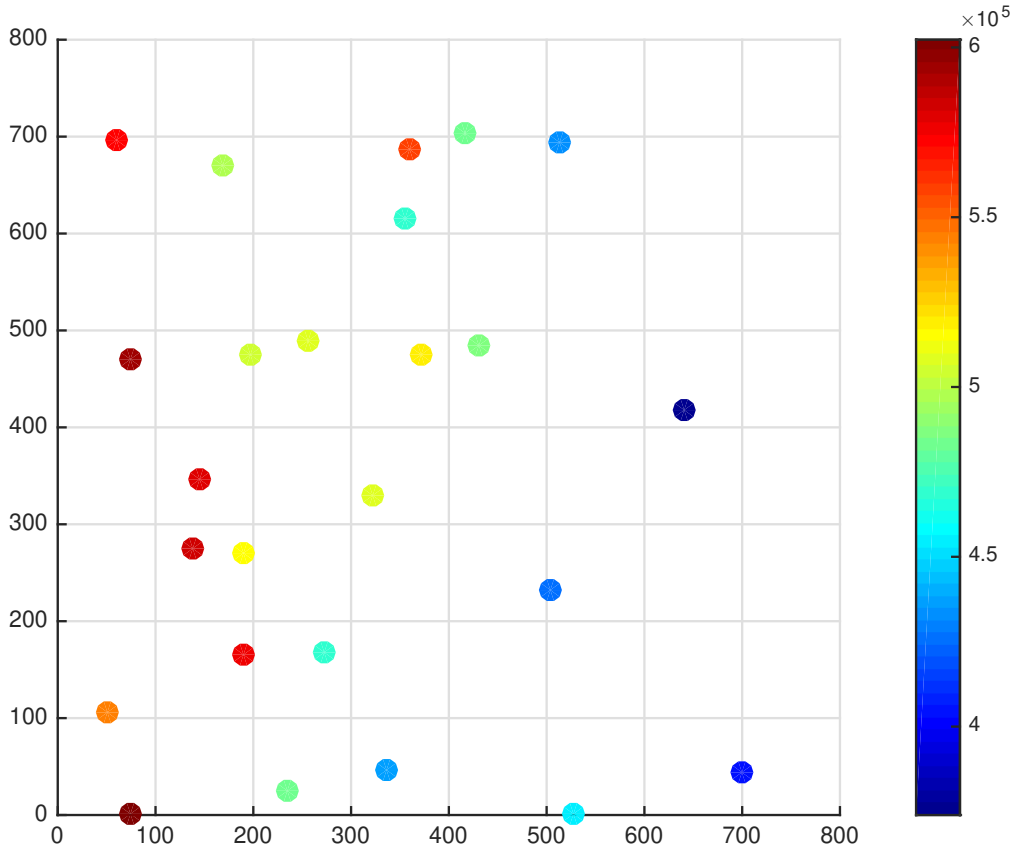


FIGURE 4.11: Best layout found for the 25 buoy study. The direction of wave propagation is from left to right. All buoys have diameter 5m, and the area is $0.707 \cdot 0.707 km^2$. The overall power out is 1.257e+7 Watts. The q-factor value is 0.9063 (initially 0.8964) and the $\overline{RCW}$ value is 1.434 (initially 1.252). The colours indicate the power generated by each buoy.

In general, our study of the interactions of each individual WEC in an array has demonstrated an example of a complex system or a multi-component system in the real-world, where solely optimising each WEC individually cannot yield the overall optimality. Moreover, our evluation of the buoy interactions were computationally prohibitively expensive, taking hours or even days. We have undertaken a problem-specific methods in order to tackle the issue. Through model approximations and caching, we achieved up to 350-fold speed-ups in the simulation times needed. This in turn allowed us to iteratively optimise the interactions in WEC arrays and to investigate this multi-component system.

As our approaches in this chapter is specific to this problem, it to some extent motivates us to explore a more general way of investigating the multi-component optimisation problems. In the next chapter, we will start to introduce our studies on the benchmark multi-component problem: the travelling thief problem.

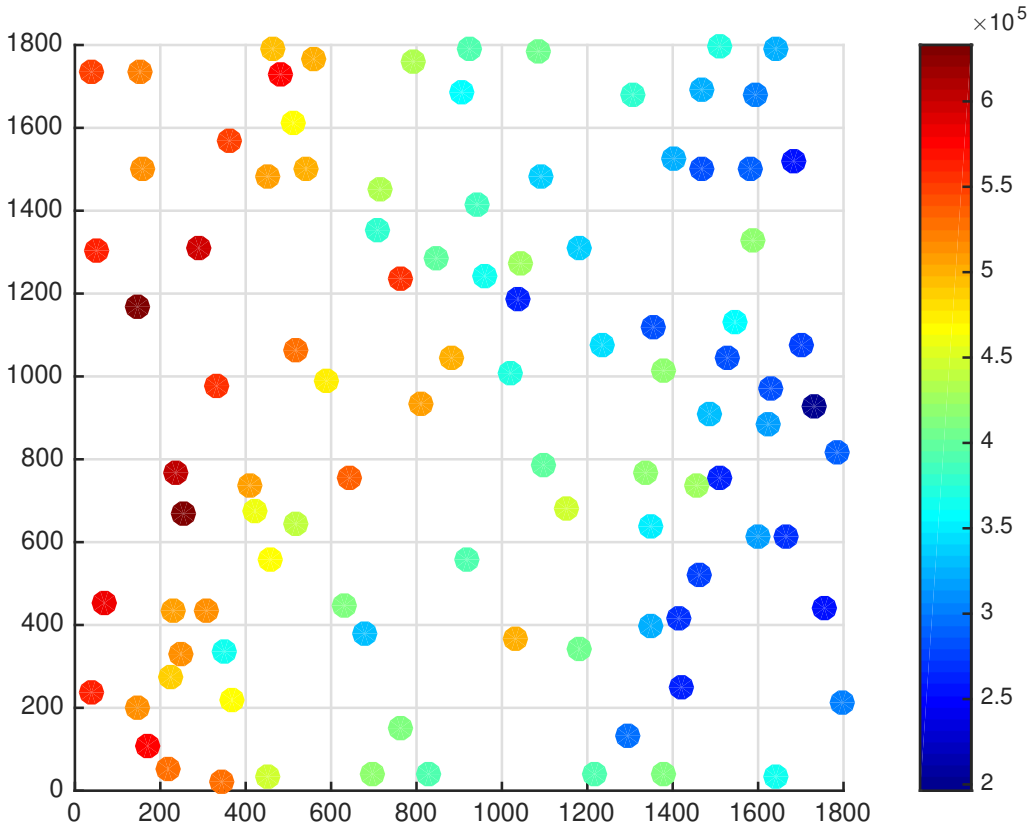


FIGURE 4.12: Best layout found for the 100 buoy study. The direction of wave propagation is from left to right. All buoys have diameter 5m, and the area is $1.8 \cdot 1.8km^2$. The overall power out is 4.147e+7 Watts. The q-factor value is 0.7476 (initially 0.6769) and the $\overline{RCW}$ value is 1.183 (initially 1.071). The colours indicate the power generated by each buoy.

Chapter 5

On the Impact of the Renting Rate for the Packing While Travelling Problem

Theoretical understanding of the TTP is not easily achievable, not only because of the interdependence of the two components but also due to the nonlinearity of the packing part. The overall gain of the packing component depends on the profit from the collected items and the transportation costs. The transportation cost nonlinearly depends on the weight of the collected items, as stated in Section 2.3 of Chapter 2. This nonlinearity distinguishes the KP component from the usual knapsack problems that are characterised by the constraint of weight (namely the capacity of the knapsack), which introduces an additional complexity. In order to understand more of this nonlinear KP component, Polyakovskiy et al. [140] propose a simplified version of the TTP: the packing while travelling (PWT) problem, as introduced in Section 2.3.4, in which the tour π is a predefined constant. Their research shows that the problem is $\mathcal{NP}$-hard even without the capacity constraint usually imposed on the knapsack (i.e. any combination of items is feasible).

On the other hand, it has been shown in [56] that good approaches for the TTP instances can be obtained by finding a near-optimal TSP tour for the underlying TSP part first, and then selecting a subset of items via a simple $(1 + 1)$ EA. This might suggest that the knapsack part is easy once a route is fixed. However, Polyakovskiy and Neumann [140] have shown that the underlying packing while travelling problem is already $\mathcal{NP}$-hard even when the capacity constraint is not imposed, i.e. any combination of items is feasible.

In this chapter, we carry out additional studies for the PWT problem. We pay special attention to an important parameter called the renting rate (R), which essentially is a factor that connects two subcomponents of the TTP. Therefore the choice of the values of this parameter may significantly influence the traits of the PWT/TTP problem. Based on the input of other parameters of the problem, such as the profits and weights of given items, we derive upper and lower bounds of R so that the simple pre-processing scheme presented in [140] cannot reduce the size of the problem instances. Furthermore, we use an evolutionary algorithm to create instances that allow us to remove as few items as possible for a fixed value of R. These studies give us additional insights into the importance of R and show the range where difficult instances for simple $(1 + 1)$ EA or the mixed-integer programming approach [140] should be found. Motivated by the success of the simple evolutionary algorithms for the packing component of the TTP, we then investigate what the difficult instances of PWT for $(1 + 1)$ EA should look like. We do this in two ways. First, we construct an instance based on the insights from rigorous runtime analysis for the classic 0-1

knapsack problem where the $(1 + 1)$ EA fails with a constant probability to obtain an optimal solution. Second, we evolve an instance for PWT using an evolutionary algorithm where the empirical failure rate of the instance is maximised.

This chapter is based on a conference paper published at the Genetic and Evolutionary Computation Conference (GECCO) [173]. In the remainder of the chapter, we explain our investigations in detail. In Section 5.1, we study the impact of the renting rate. Section 5.2 proposes a mechanism to increase the hardness of instances. The theoretical elements and the evolutionary algorithm for finding hard PWT instances are given in Section 5.3. Finally, we draw a conclusion in Section 5.4.

5.1 Impact of the Renting Rate

As stated in Section 2.3.4 of Chapter 2, the PWT problem can mainly be defined as

$$\mathcal{Z}_\pi(\rho) = \mathcal{P}(\rho) - R \cdot \mathcal{T}(\rho), \tag{5.1}$$

where $\mathcal{P}(\rho)$ is a total profit of selected items and $\mathcal{T}(\rho)$ is a total travelling time. We let $\mathcal{T}(\rho) = \mathcal{T}_\pi(\rho)$ for the sake of conciseness. It can be seen here that the renting rate R has significant impact on the overall benefit. From one hand, if R is small enough (i.e. $R \to 0$), then $R \cdot \mathcal{T}(\rho) \to 0$, which implies that the total transportation cost of any set of items becomes negligible. Specifically, for any value of R chosen from $[0, R_L)$, the optimal solution is to select all the items, i.e. $\rho = \{1\}^m$. On the other hand, when $R \to \infty$, a solution tends to have no item selected. So, for any value of R chosen from $(R_U, \infty]$ the solution $\rho = \{0\}^m$ becomes optimal. Let the interval (R_L, R_U) be named as a *non-trivial range* of the renting rate. Then we use it to search for hard instances of PWT. Indeed, any value of R other than those in the interval (R_L, R_U) makes the problem trivial. In this sense, R_L and R_U act as a lower and an upper bound on R, respectively. Our work shows that (i) hard PWT instances should be found when $R \in (R_L, R_U)$ and (ii) a ratio related to R named as a *non-trivial items rate* may be used to evaluate the hardness of instances. In the rest of sections in this chapter, we elaborate on the both propositions.

5.1.1 General Bounds

The solution $\rho = \{1\}^m$ means the situation when the profit of any item $e_{\theta k} \in M$ is large enough to cover the cost of its transportation from city θ to the last city $n + 1$ along with any possible subset of items. That is $p_{\theta k} \geq R \cdot (\mathcal{T}(S) - \mathcal{T}(S \setminus \{e_{\theta k}\}))$ for any item $e_{\theta k} \in S$ and for any subset of items $S \subseteq M$. In order to find lower bound R_L, we define an auxiliary problem where R needs to be maximised:

$$\begin{aligned} &R_L = \max R \\ &\text{s.t. } p_{\theta k} \geq R \cdot (\mathcal{T}(S) - \mathcal{T}(S \setminus \{e_{\theta k}\})),\ \forall e_{\theta k} \in S,\ \forall S \subseteq M \end{aligned} \tag{5.2}$$

In (2.12), v_i depends linearly on the weight of collected items while each of the travelling times contributing to (2.15) depends inversely on v_i. This gives us $(\mathcal{T}(M) - \mathcal{T}(M \setminus \{e_{\theta k}\})) \geq (\mathcal{T}(S) - \mathcal{T}(S \setminus \{e_{\theta k}\}))$ as shown in [140]. Therefore, (5.2)

can be replaced by $p_{\theta k} \geq R \cdot (\mathcal{T}(M) - \mathcal{T}(M \setminus \{e_{\theta k}\}))$, which results in a bound on R as follows:

$$R \leq \frac{p_{\theta k}}{\mathcal{T}(M) - \mathcal{T}(M \setminus \{e_{\theta k}\})} \tag{5.3}$$

We now consider the worst case scenario by assuming that all the items are located in the first city and have the maximum possible weight. Therefore, to derive R_L, we set $\theta = 1$, $p_{\theta k} = P_L$, and $w_{ik} = W_U$ for any item $e_{ik} \in M$. This setting corresponds to the smallest value of the right-hand side in (5.3), where $\mathcal{T}(M)$ and $\mathcal{T}(M \setminus \{e_{\theta k}\})$ get the form:

$$\mathcal{T}(M) = \sum_{i=1}^{n} \frac{d_i}{v_{max} - \nu m W_U}$$
$$\mathcal{T}(M \setminus \{e_{\theta k}\}) = \sum_{i=1}^{n} \frac{d_i}{v_{max} - \nu (m-1) W_U}$$

Finally, R_L is computed as follows:

$$R_L = P_L \Big/ \sum_{i=1}^{n} \frac{d_i \nu W_U}{(v_{max} - \nu m W_U)(v_{max} - \nu (m-1) W_U)} \tag{5.4}$$

We proceed similarly to find R_U. In fact, the solution $\rho = \{0\}^m$ means the situation when the profit of any item $e_{\theta k} \in M$ is too small to cover its transportation cost from city θ along with any subset of items. That is $p_{\theta k} \leq R \cdot (\mathcal{T}(S \cup \{e_{\theta k}\}) - \mathcal{T}(S))$ for any item $e_{\theta k} \in M$ and for any subset of items $S \subseteq M \setminus \{e_{\theta k}\}$. In order to find the upper bound R_U, we define an auxiliary problem where R is to be minimised:

$$\begin{aligned} &R_U = \min R \\ &\text{s.t. } p_{\theta k} \leq R \cdot (\mathcal{T}(S \cup \{e_{\theta k}\}) - \mathcal{T}(S)),\ \forall e_{\theta k} \in M,\ \forall S \subseteq M \setminus \{e_{\theta k}\} \end{aligned} \tag{5.5}$$

Given that $(\mathcal{T}(\{e_{\theta k}\}) - \mathcal{T}(\emptyset)) \leq (\mathcal{T}(S \cup \{e_{\theta k}\}) - \mathcal{T}(S))$ in [140], we replace (5.5) by $p_{\theta k} \leq R \cdot (\mathcal{T}(\{e_{\theta k}\}) - \mathcal{T}(\emptyset))$, which bounds R as follows:

$$R \geq p_{\theta k} \Big/ \sum_{i=\theta}^{n} d_i \left(\frac{1}{v_{max} - \nu w_{\theta k}} - \frac{1}{v_{max}} \right) \tag{5.6}$$

Here, we consider the best case scenario by assuming that item $e_{\theta k}$ is placed into city n, i.e. $\theta = n$, and has $p_{\theta k} = P_U$ and $w_{\theta k} = W_L$. This corresponds to the maximum value of the right-hand side in (5.6). Finally, R_U is to be computed as follows:

$$R_U = \frac{P_U v_{max} (v_{max} - \nu W_L)}{\nu W_L d_n} \tag{5.7}$$

As an example, let us consider the instance `eil51` with 51 cities from TSPLIB [145]. We assume that each but the last city contains 5 items with weights and profits bounded by $P_L = W_L = 1$ and $P_U = W_U = 1000$, respectively. We also set $v_{max} = 1$

n:	50	m:	250
P_L:	1	P_U:	1,000
W_L:	1	W_U:	1,000
v_{max}:	1	v_{min}:	0.1
C:	250,000	ν:	3.60e−06

TABLE 5.1: Input parameters for generating PWT instances

and $v_{min} = 0.1$ to limit the velocity of the thief, and set capacity $C = 250,000$ so that all the items can be selected. The set of input parameters to generate the PWT instance is given in Table 5.1. Having the route as shown in Table 5.2, we calculate the bounds on R to be $R_L = 6.85\text{e}{-2}$ and $R_U = 4.63\text{e}{+7}$ according to equations (5.4) and (5.7). However, being restricted to these bounds, the non-trivial range looks tremendous without much guarantee whether the instance is easy or hard to solve.

Route#	From	To	Distance	Route#	From	To	Distance
1	1	22	7	26	42	19	9
2	22	8	12	27	19	40	11
3	8	26	7	28	40	41	12
4	26	31	10	29	41	13	9
5	31	28	6	30	13	25	13
6	28	3	9	31	25	14	6
7	3	36	12	32	14	24	11
8	36	35	6	33	24	43	12
9	35	20	7	34	43	7	12
10	20	2	12	35	7	23	6
11	2	29	9	36	23	48	9
12	29	21	7	37	48	6	9
13	21	16	10	38	6	27	9
14	16	50	6	39	27	51	8
15	50	34	6	40	51	46	2
16	34	30	7	41	46	12	7
17	30	9	8	42	12	47	6
18	9	49	6	43	47	18	8
19	49	10	8	44	18	4	8
20	10	39	10	45	4	17	8
21	39	33	14	46	17	37	5
22	33	45	7	47	37	5	11
23	45	15	7	48	5	38	7
24	15	44	6	49	38	11	7
25	44	42	10	50	11	32	6

TABLE 5.2: A tour generated by Lin-Kernighan TSP heuristic [101] for the `Eil51` instance of TSPLIB

5.1.2 Item-Specific Bounds

We further investigate how the value of renting rate R within the non-trivial range influences the hardness of the problem. Now, instead of focusing on the range of R generally, we investigate whether each item $e_{\theta k} \in M$ may have its own item-specific non-trivial range. If such an item-specific range exists for each of the items, we calculate the rate between the number of items whose range includes the value of R and the total number of items m. We then use this rate as a measure of hardness. In general, item $e_{\theta k}$ being trivial means that it should be either selected or discarded (i.e. as a compulsory or unprofitable item [140]). Specifically, we define item $e_{\theta k}$ as a *non-trivial item* if there is a range $(R^L_{\theta k}, R^U_{\theta k})$ such that $R \in (R^L_{\theta k}, R^U_{\theta k})$. According to Propositions 1 and 2 in [140], we calculate the range $(R^L_{\theta k}, R^U_{\theta k})$ of $e_{\theta k}$ as follows.

$$R^L_{\theta k} = p_{\theta k}/(\mathcal{T}(M) - \mathcal{T}(M \setminus \{e_{\theta k}\})) \tag{5.8}$$

$$R^U_{\theta k} = p_{\theta k}/(\mathcal{T}(\{e_{\theta k}\}) - \mathcal{T}(\emptyset)) \tag{5.9}$$

We can now define *non-trivial items rate* as follows.

Definition 1. *Non-trivial items rate λ is a fraction of the number of non-trivial items in the total number of items m in an instance.*

Figure 5.1 illustrates the item-specific non-trivial ranges for the items of the instance presented in Table 5.2. Here, we assume that each city but the last one contains a single item only with a profit and a weight as uniformly random values in $[P_L, P_U]$ and $[W_L, W_U]$, respectively. The non-trivial ranges of the items are represented by bars. Interestingly, the bars are distributed logarithmically in the figure.

Now, to compute the non-trivial items rate, one needs to set a particular value for R. It can be done by selecting R corresponding to one of the vertical lines in the Figure 5.1. Usually, such a line only crosses some of the bars. For example, assuming $R = 10^{+2}$, we have 19 out of 50 items having their bars crossed by the related line. In fact, those items whose bars are crossed by the line are non-trivial. The non-trivial items rate λ therefore equals 0.38 in this case. Similarly, if we set $R = 10^{+4}$, then λ becomes 0.02.

Observing the Figure 5.1 further, one can see that a better non-trivial items rate λ can be obtained by setting R to be around $2.0\text{e}{+2}$. However, it still remains below 0.5. In fact, finding the high values of λ, for example 0.8, is not possible for this instance. Indeed, we would like to have instances with λ as large as possible, because such instances are intuitively hard to solve than ones with low λ. We therefore design a simple evolutionary algorithm to generate instances with maximised non-trivial items rate.

5.2 Maximising Non-Trivial Items Rate

In order to obtain PWT instances with large non-trivial items rate λ, we propose a simple evolutionary algorithm named λEA, in which each weight or profit value is mutated with probability $1/m$. In each iteration, the selection procedure of λEA chooses an offspring instance as the best solution if its corresponding λ is larger than one that the parent solution has. λEA runs mutation and selection operations

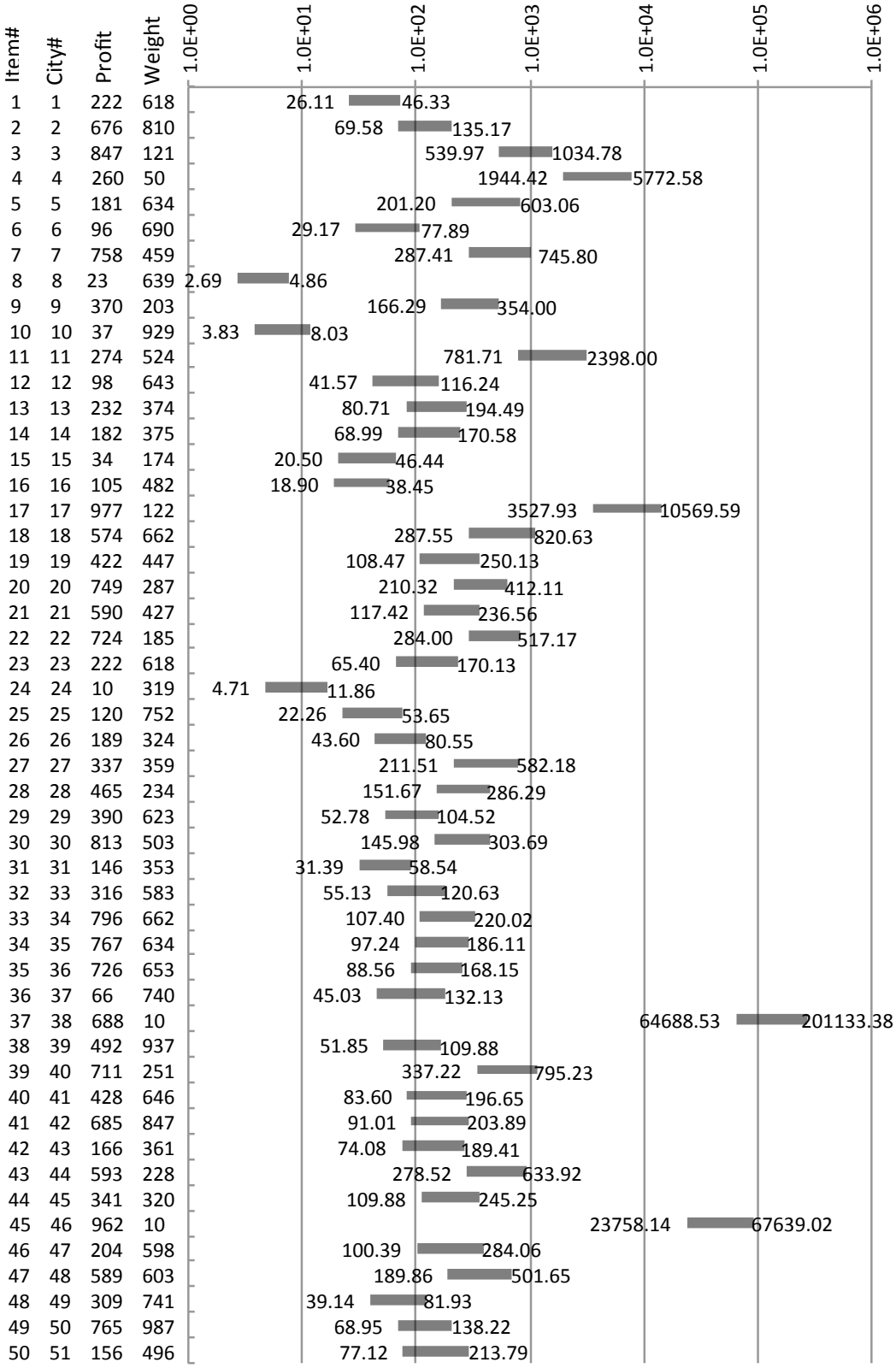


FIGURE 5.1: Illustrating the non-trivial ranges of the items for the instance of Table 5.2

iteratively until the limit on the number of iterations is reached. The detailed pseudocode of λEA is listed in Algorithm 10.

We use the same TSP instance described in Table 5.2 with the input parameters provided in Table 5.1 to illustrate the work of our algorithm. In order to understand how the particular values of renting rate R within the non-trivial range influence the non-trivial items rate, we run each experiment on a given set of $R \in \{10^k : k \in \{-3, -2, ..., 8\}\}$, which covers the known non-trivial range $(6.85\text{e}{-2}, 4.63\text{e}{+7})$ calculated in the previous section. We rerun the algorithm 5 times for each specific value of R setting the maximal number of iterations to 100,000.

The results of λEA are presented in Figure 5.2. The graph corresponding to the different values of the non-trivial items rate obtained by λEA is drawn by red. For a comparison purpose, we use the result of random generation of items. Specifically, we generate one million random instances for each of the values of R and select the instance with maximal λ. The corresponding graph is drawn by blue. The horizontal axis of Figure 5.2 represents the particular values of the renting rate. In the red graph, each point is provided with a bar depicting the standard deviation of the non-trivial items rate. The vertical axis of the figure represents the resulting values of λ.

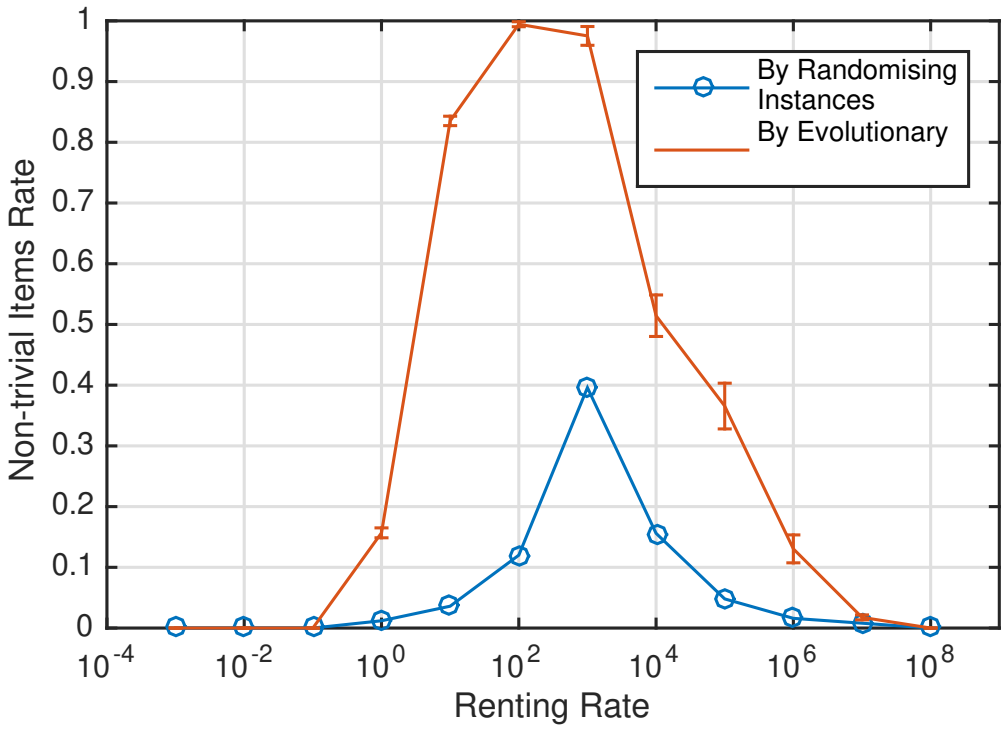


FIGURE 5.2: Non-trivial items rate computed by λEA

According to the Figure 5.2, λ is strongly positive for both λEA and the random generator when $R \in \{1, ..., 10^7\}$. By contrast, the results for $R \in \{10^{-3}, 10^{-2}, 10^{-1}\}$ and $R \in \{10^{+7}, 10^{+8}\}$ are very close to zero. When $R \in \{1, ..., 10^{+6}\}$, the values of λ obtained by λEA are significantly larger than those produced by the random generator, yet the standard deviation is small.

Algorithm 10 λEA to maximise the non-trivial items rate

procedure λEA(Maximal number of iterations)
    INITIALISATION
    repeat
        SELECTION
        MUTATION
    until the maximal number of iterations is reached
    Exit
end procedure
procedure INITIALISATION
    Initialise an PWT instance with a set of items M
    Set the value of renting rate R
    Set $p_{ik} \in [P_L, P_U]$ and $w_{ik} \in [W_L, W_U]$ uniformly at random for each item $e_{ik} \in M$
    λ = FITNESS(initial instance)
end procedure
procedure MUTATION
    for each item $e_{ik} \in M$ **do**
        Set $b \in [0, 1]$ uniformly at random
        if $b \leq \frac{1}{m}$ **then**
            Set $w_{ik} \in [W_L, W_U]$ uniformly at random
        end if
        Set $b \in [0, 1]$ uniformly at random
        if $b \leq \frac{1}{m}$ **then**
            Set $p_{ik} \in [P_L, P_U]$ uniformly at random
        end if
    end for
end procedure
procedure SELECTION
    λ' = FITNESS(new instance)
    if $\lambda' > \lambda$ **then**
        Choose the new instance
    end if
end procedure
function FITNESS(instance)
    Calculate non-trivial items rate λ
    return λ
end function

Establishing a threshold for λ, say 0.8, we see that only some values of R, i.e. 10^{+1}, 10^{+2} and 10^{+3}, produce those values of λ which are expected to give hard instances. In fact, the range of R producing high values of λ is much smaller than the general non-trivial range. When λEA reaches large values of λ, the item-specific ranges of items become close or they include the predefined value of R. This can be seen through the comparison of Figures 5.3(a) and 5.3(b). Specifically, Figure 5.3(a) depicts the initial random instance for which λ is only equal to 0.12. In this case, the item-specific ranges are widely distributed around the line corresponding to $R = 10^{+2}$. The results of λEA are shown on the Figure 5.3(b), where the best found instance has $\lambda = 1$, i.e. all the items are non-trivial. It gets all the item-specific ranges concentrated around the line corresponding to $R = 10^{+2}$.

Our computational experiments presented in Section 5.3.2 confirm that λEA algorithm produces hard instances for PWT. In fact, more items are getting the value of R included into their item-specific ranges, the less items can be excluded by the pre-processing in [140] and the harder the instance becomes. In other words, it turns out to be hard to define which items are to be selected in the optimal solution when most of the items contain the value of R in their item-specific ranges.

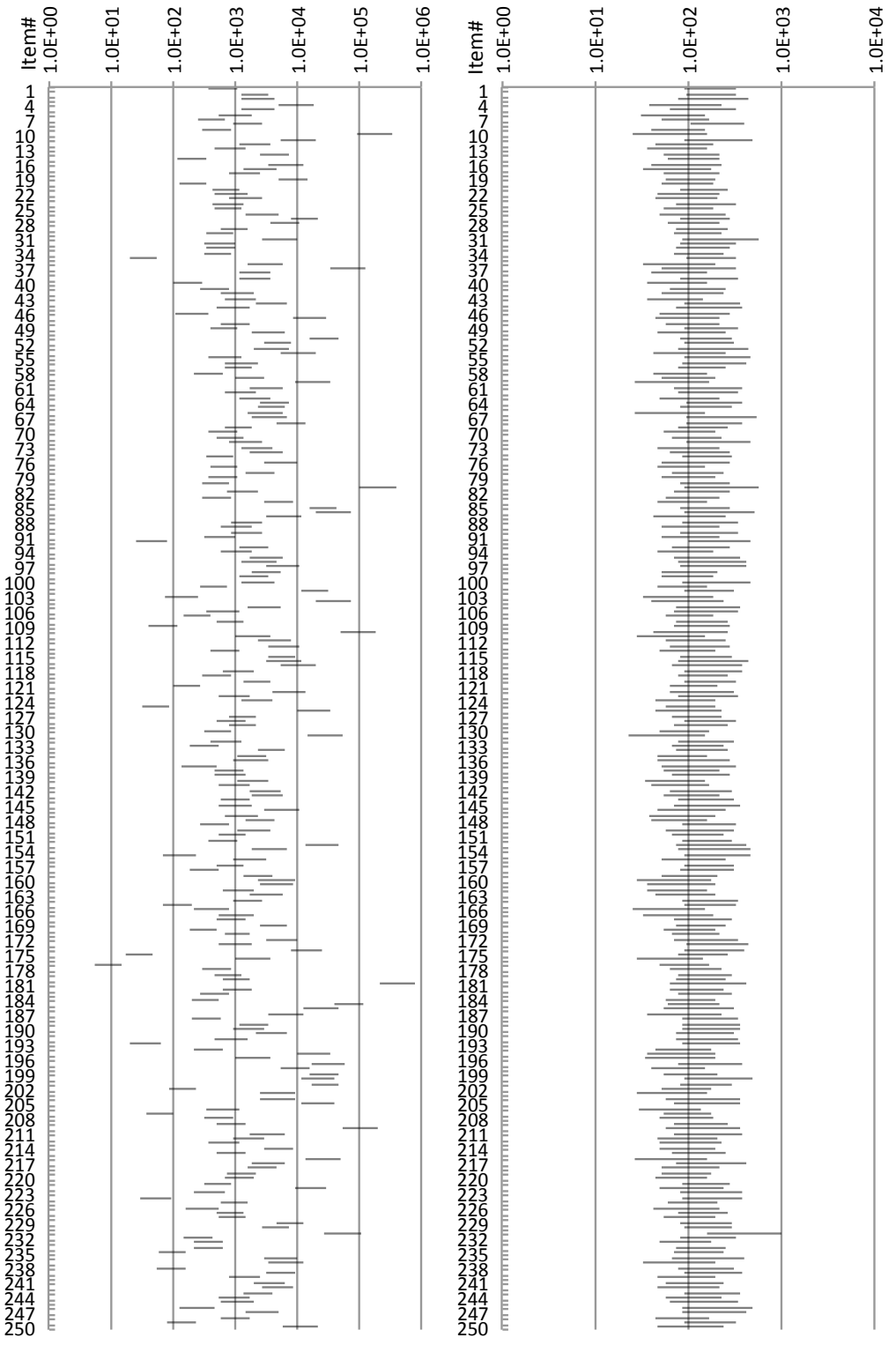
5.3 Hard Instances for the (1+1) EA

We now carry out theoretical and experimental investigations for the classical $(1+1)$ EA on instances of the packing while travelling problem. Our goal is to provide additional insights on which instances are hard to be solved by simple evolutionary algorithms. The $(1+1)$ EA is a standard algorithm investigated in the area of runtime analysis of evolutionary computation [6, 130], and therefore well suited for understanding of working behaviour of evolutionary computing techniques for PWT. The description of the $(1+1)$ EA considered in this section is given in Algorithm 11. It starts with a solution chosen uniformly at random and uses mutation flipping each bit with probability $1/m$ in order to produce an offspring. Being not worse than it's parent according to the fitness function, the offspring replaces it.

5.3.1 Theoretically Constructed Instance

In this section, we present an PWT problem instance where $(1+1)$ EA fails with a constant probability. The instance is motivated by an instance for the classical knapsack problem for which it has been shown that the (1+1) EA has an exponential expected optimisation time [180]. Furthermore, it has been used for investigations of utility functions in the area of evolutionary multi-objective optimisation [128].

Our instance consists of two cities only. The distance between the cities is set to 1. The first city contains $m = r + 1$ items while the second city is a destination point free of items. We give r items of the first city the same profit and weight such that cost $c_k = p_{1k} = w_{1k} = 1$, $1 \leq k \leq r$. Similarly, item $m = r + 1$ gets its profit and weight as cost $c_m = p_{1m} = w_{1m} = r + 1$. We call the former items *simple* and the later one *unique*. To establish the unconstrained settings, we specify $C = 2r + 1$. Subsequently, we set $v_{max} = 2$ and $v_{min} = 1$ which implies $\nu = 1/C$. Finally, we define $R^* = C \cdot (2 - (r+1)/C)^2$.



(a) Initial instance ($\lambda = 0.12$) (b) The best instance found by λEA ($\lambda = 1$)

FIGURE 5.3: Comparing the distribution of item-specific ranges of the instance generated randomly and one obtained by λEA

Algorithm 11 $(1+1)$ EA to solve an PWT instance

```
1: procedure INITIALISATION
2:     Generate ρ ∈ {0,1}^m uniformly at random
3: end procedure
4: repeat
5:     procedure MUTATION
6:         for each y_ik in ρ do
7:             Set b ∈ [0,1] uniformly at random
8:             if b ≤ 1/m then
9:                 Flip the value of y_ik
10:            end if
11:        end for
12:    end procedure
13:    procedure SELECTION
14:        Select the instance with better result
15:    end procedure
16: until the maximal number of iterations is reached
```

In order to show why the $(1+1)$ EA fails to find the optimal solution in polynomial time, we use arguments similar to ones discussed in [180, 128]. The basic idea is that with probability $1/2$ the item $m+1$ is not included into the population. Furthermore, the expected number of simple items in the initial solution is $r/2$ and at least $(1/2-\epsilon)r$, $\epsilon > 0$ a constant, with probability $1-e^{-\Omega(m)}$ using Chernoff bounds. The waiting time for such a step is exponentially small in the number of bits that have to be flipped. This implies an exponential optimisation time of the (1+1) EA when starting with an initial solution that has not chosen item $r+1$ and many simple items.

We relate the previous ideas to our instance of PWT. Taking into account the given properties, we now specify the objective function (2.13) as

$$f_{R^*}(w) = w - \frac{R^*}{2 - w/C}, \tag{5.10}$$

where the input $w = \sum_{k=1}^{r+1} c_k y_k$ is restricted to discrete integer values respecting the packing plan $\rho \in \{0,1\}^m$. When defined on the interval $[0, C]$, $f_{R^*}(w)$ reaches its unique maximum in the point $w^* = C \cdot (2 - \sqrt{R^*/C}) = r+1$. In our particular settings, achieving $w^* = r+1$ is possible when the unique item is solely selected in the packing plan ρ. In other words, selecting item $r+1$ only results in the unique global optima for the given class of instances.

For a given real $\epsilon \geq 0$, we can establish that $f_{R^*}(w) > f_{R^*}(w+r+1)$ holds for $w \geq (1/2-\epsilon)\, r$ starting with a certain value of r, $r \to \infty$. This implies that flipping the bit corresponding to item $r+1$ is not accepted. To give a formal proof for the $(1+1)$ EA we would need to consider multiple bit flips in addition. This can be done by a more detailed analysis.

5.3.2 Evolving Hard Instances

In this section, we study one more qualitative method to produce and evaluate hard instances for PWT. Indeed, the running time of an algorithm is one of traditional ways to measure hardness of instances. It is applicable for exact approaches like the MIP approach introduced by Polyakovskiy et al. [140]. However, for our aims we are interested in a method that provides additional insights on which instances are hard to be solved by $(1+1)$ EA. We introduce a *failure rate* δ that measures the number of times $(1+1)$ EA fails to obtain a result which is better than one with a predefined objective value. Therefore, we propose another evolutionary algorithm (δEA) to search for hard instances with a given renting rate. It is similar to λEA described in Algorithm 10 with only a difference that the failure rate measure is employed instead of the non-trivial items rate as a fitness procedure. Algorithm 12 sketches the fitness function used to compute δ in δEA.

Algorithm 12 Fitness function to calculate failure rate δ

```
1:  function FITNESS(an PWT instance)
2:      Call the MIP approach to solve the instance
3:      Set counter i = 0
4:      for 20 times do
5:          Call the (1 + 1) EA to solve the instance
6:          if the result of the (1 + 1) EA is worse than the one of the MIP approach
    then
7:              Increment counter i = i + 1
8:          end if
9:      end for
10:     return i / 20
11: end function
```

To calculate failure rate δ, δEA employs the solution returned by the the exact MIP approach [140] to set the desired objective value, i.e. the total benefit. It applies $(1+1)$ EA to solve an PWT instance as given in Algorithm 11. Specifically, the fitness function restarts $(1+1)$ EA 20 times setting the maximum number of iterations to be 20,000. Each time the outcome of the $(1+1)$ EA is worse than the result of the MIP approach, the function counts it as a fail. It then calculates the failure rate as the fraction of the total number of fails in the total number of runs.

We use the same values of R as defined in Section 5.2 so that the results of δEA and λEA can be compared. We run δEA 5 times for each given value of R setting the maximal number of iterations to be 10,000. Figure 5.4 plots the failure rates obtained, while Figures 5.5 and 5.6 show the average runtimes of the MIP approach and ones of the $(1+1)$ EA, respectively. All the three figures have bars to illustrate the average values and standard deviations. The horizontal axes for each of the three have identical renting rate same to the one in the Figure 5.2 for λEA. In such a method, we can study whether and how changing the value of renting rate R may influence the hardness of the instance to be solved by $(1+1)$ EA, as we assume that the larger the failure rate becomes, the harder the instance is. In addition, we expect that our assumption can be supported by the running time of the exact MIP approach.

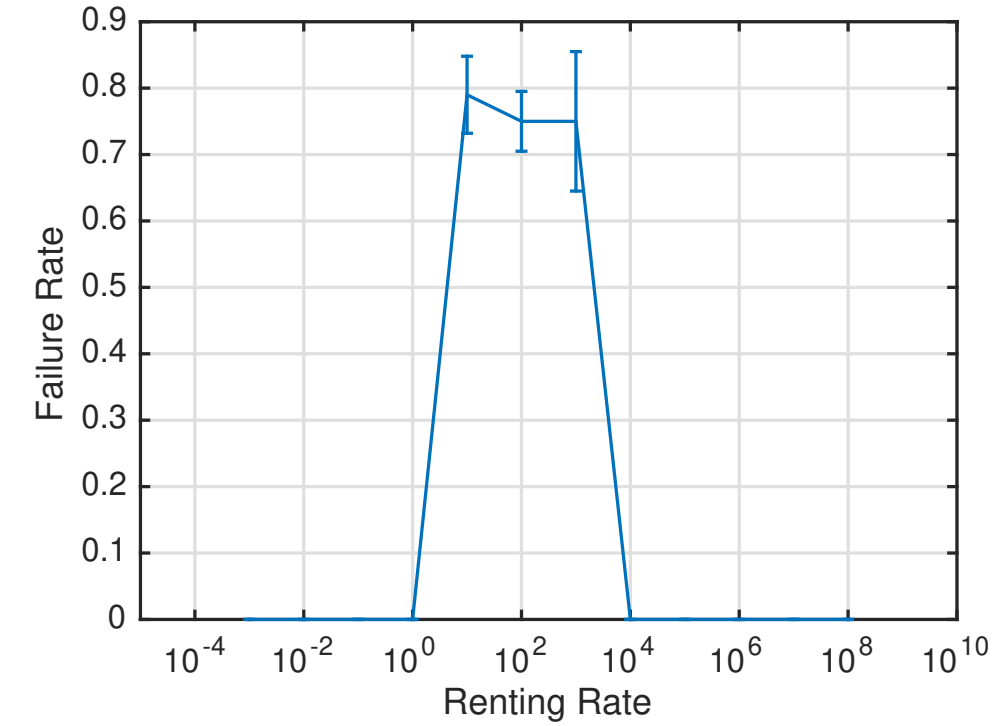


FIGURE 5.4: Failure rate δ of $(1 + 1)$ EA

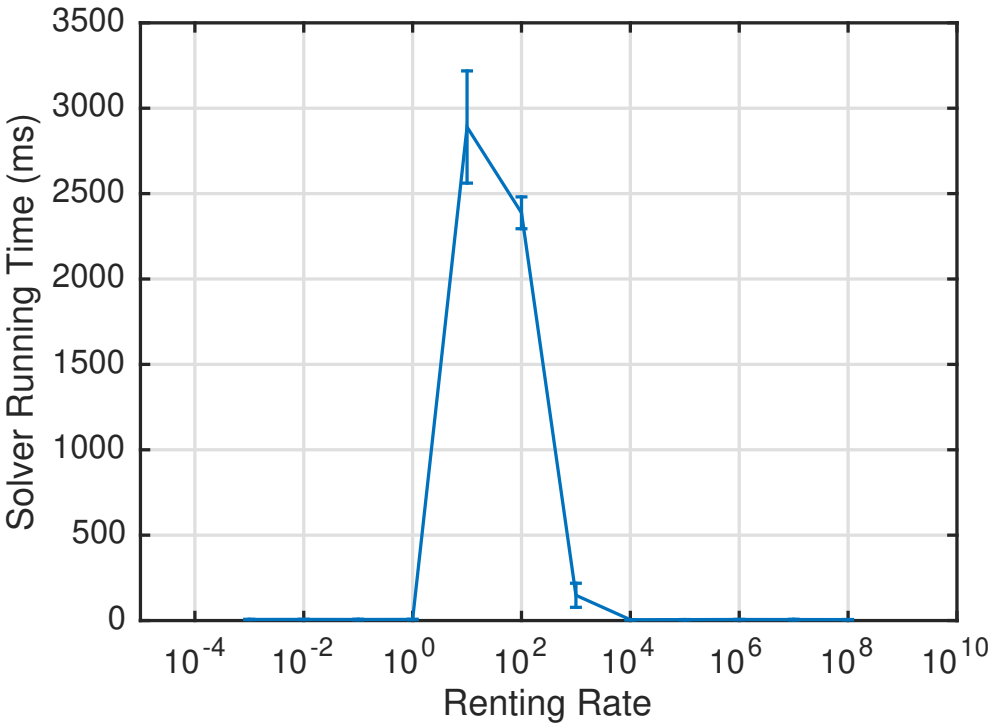


FIGURE 5.5: Runtime (ms) of the MIP approach

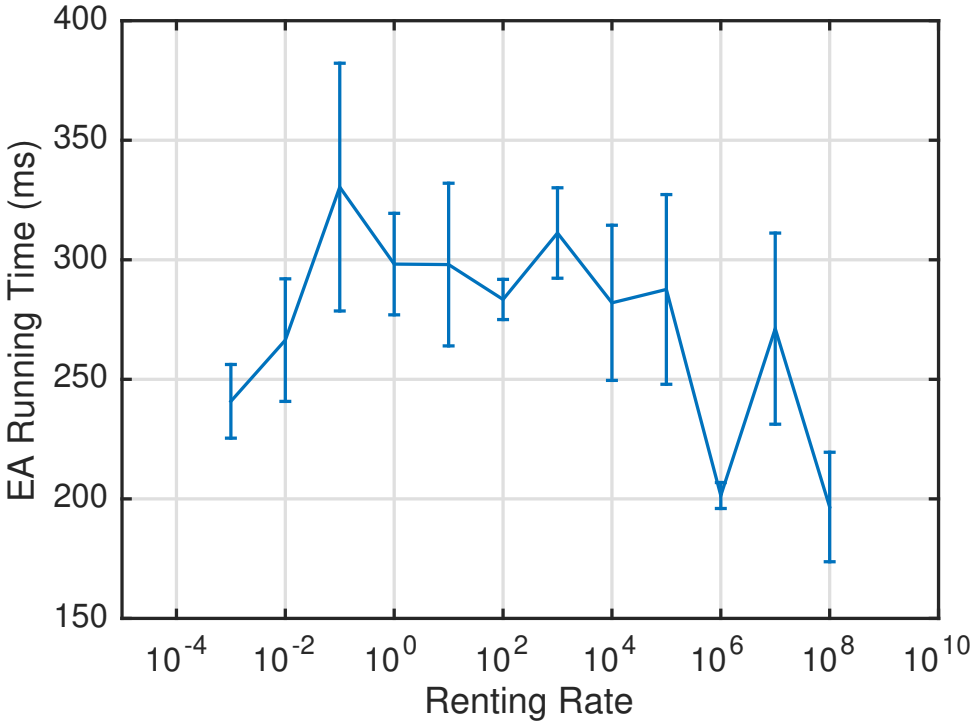


FIGURE 5.6: Runtime (ms) of the $(1 + 1)$ EA

In Figure 5.4, one can observe that only 3 out of 12 values of R, i.e. $\{10^1, 10^2, 10^3\}$ have the values of the failure rate greater than zero. This exactly matches the results of λEA in Section 5.2, where instances with high λ have been found for the same values of R. In fact, δEA finds instances with large failure rate δ when λEA does this for large values of λ. Moreover, Figure 5.5 reports that runtimes of the MIP approach are significantly larger when R takes one of these three values. This implies that hard instances for the MIP approach correspond to these values of R. In contrast, the running time of the $(1 + 1)$ EA solver seems not to be much influenced by the hardness of an instance, which might be partially because that the $(1 + 1)$ EA is terminated according to the maximal number of iterations instead of the convergence.

The results of Figure 5.4 and 5.5 to some extents confirm our conjecture that the values of the non-trivial items rate λ may be used as an indicator of hard and easy instances. In practice, we may calculate the non-trivial range of R first and then use λEA to explore it to search for largest values of λ with respect to particular values of R. Furthermore, one may analyse a set of instances with respect to the value of λ. Then a threshold on the value of λ may be defined and employed as a mechanism to eliminate easy instances from a benchmark.

5.4 Conclusion

Through this chapter, we contributed to the knowledge and understanding of multi-component combinatorial optimisation problems that arise frequently in real-world applications, such as logistics and supply chain management. We investigated the packing while travelling (PWT) problem, motivated by the recently-introduced the travelling thief problem (TTP) and studied the impact of the renting rate, which connects the profit and the transportation costs of the problem. We have shown through theoretical and experimental investigations, how the renting rate affects the number of items that can be eliminated by the simple pre-processing scheme and therefore the hardness of the particular instances. Furthermore, we have constructed instances in a theoretical and experimental way, where a simple baseline $(1 + 1)$ EA fails to obtain an optimal solution. In the next chapter, we will continue our study of the PWT problem in terms of the exact approach and corresponding approximation.

Chapter 6

A Dynamic Programming and Corresponding Fully Polynomial Time Approximation Scheme for the Packing While Traveling Problem

In the previous chapter (Chapter 5) we have conducted a study on the packing while travelling (PWT) problem, in which we show that by adjusting a single parameter we can influence the overall hardness of a PWT/TTP instance. In this chapter, we continue our investigations into the PWT problem. We introduce a dynamic programming approach and the corresponding fully polynomial time approximation scheme for it.

The key idea of our dynamic programming is to consider the items in the order they appear on the route that needs to be travelled and apply dynamic programming similar to the process undertaken for the classical knapsack problem (KP, Section 2.3.2). When considering an item, the decision has to be made as to whether or not to pack the item. The dynamic programming approach computes for the first i, $1 \leq i \leq m$, items, and for each possible weight w, the maximal objective value that can be obtained. As the programming table that is used depends on the number of different possible weights, the algorithm runs in pseudo-polynomial time.

After having obtained the exact approach based on dynamic programming, we consider the design of a fully polynomial approximation scheme (FPTAS) [75]. We designed an FPTAS for the amount that can be gained over the travel costs, when the vehicle travels empty (which is the minimal possible travel cost). Our FPTAS makes use of the observation that the item with the largest reward leads to an objective value of at least OPT/m and uses appropriate rounding in the previously-designed dynamic programming approach.

We evaluate our two approaches on a wide range of instances from the TTP benchmark set [139] and compare them with the exact and approximative approaches given in [140]. Our results show that a large majority of the instances that can be handled by exact methods, are actually solved much more quickly by dynamic programming than the previously-developed mixed integer programming and branch-infer-and-bound approaches. Considering instances with a larger profit and weight range, we show that the choice of the approximation guarantee significantly impacts the runtime behaviour.

This chapter is based on a paper submitted to a journal [129]. The remainder is structured as follows: we present the exact dynamic programming approach in

Section 6.1 and design an FPTAS in Section 6.2. Our experimental results are shown in Section 6.3. Finally, we finish this chapter with some conclusions drawn from our investigations.

6.1 Dynamic Programming

We introduce a dynamic programming approach for solving the PWT defined in Section 2.3.4. Dynamic programming is one of the traditional approaches for the classical 0-1 knapsack problem [165], which composes a 2-dimensional table consisting of at most W columns and m rows to compute the optimality. In the 0-1 knapsack problem, W is the capacity of the knapsack and m is the number of items. Similarly in our problem, W is either the capacity of the knapsack or the total weight of all items whichever is smaller, i.e. $W = \min(C, \sum_{i=1}^{n} \sum_{j=1}^{m_i} w_{ij})$, and m is the number of items as well. Items are processed in the order they appear along the path N and we consider them in the lexicographic order with respect to their indices, i.e.

$$e_{ab} \preceq e_{ij}, \text{ iff } ((a < i) \vee (a = i \wedge b \leq j)).$$

Note that $\preceq$ is a total strict order and we process the items in this order starting with the smallest element. The entry $\zeta_{i,j,k}$ represents the maximal reward that can be obtained by considering all combinations of items e_{ab} with $e_{ab} \preceq e_{ij}$ leading to weight exactly k. We let $\zeta(i, j, \cdot)$ denote the column containing the entries $\zeta_{i,j,k}$. In the case that a combination of weight k doesn't exist, we set $\zeta_{i,j,k} = -\infty$. We let

$$d_{in} = \sum_{l=i}^{n} d_l$$

denote the distance from city i to the last city $n + 1$.

We let $\mathcal{Z}(\emptyset)$ denote the reward of the empty set which is equivalent to the travel cost when the vehicle travels empty. Furthermore, $\mathcal{Z}(e_{ij})$ denotes the reward when only item e_{ij} is chosen.

For the first item e_{ij} according to $\preceq$, we set

$$\zeta(i, j, 0) = \mathcal{Z}(\emptyset),$$

$$\zeta(i, j, w_{ij}) = \mathcal{Z}(e_{ij}),$$

and

$$\zeta(i, j, k) = -\infty \text{ iff } k \notin \{0, w_{ij}\}.$$

Let $e_{i'j'}$ be the predecessor of item e_{ij} in $\preceq$. Based on $\zeta(i', j', \cdot)$ we compute for $\zeta(i, j, \cdot)$ each entry $\zeta_{i,j,k}$ as

$$\max \begin{cases} \zeta_{i',j',k} \\ \zeta_{i',j',k-w_{ij}} + p_{ij} - Rd_{in}\left(\frac{1}{v_{max} - \nu k} - \frac{1}{v_{max} - \nu(k - w_{ij})}\right) \end{cases}$$

Let e_{st} be the last element according to $\preceq$, then $\max_k \zeta(s,t,k)$ is reported as the value of an optimal solution. We now investigate the runtime for this dynamic program. If d_{in} has been computed for each i, $1 \leq i \leq n-1$, which takes $O(n)$ time in total, then each entry can be compute in constant time.

To speed up the computation of our DP approach, we only store an entry for $\zeta(i,j,k)$ if it is not dominated by any other entry in $\zeta(i,j,\cdot)$, i.e. there is no other entry $\zeta(i,j,k')$ with $\zeta(i,j,k') \geq \zeta(i,j,k)$ and $k' < k$. This does not affect the correctness of the approach as an item e_{ij} can be added to any entry of $\zeta(i',j',\cdot)$ and therefore we obtain for each dominated entry at least one entry in the last column having at least the same reward but potentially smaller weight. The algorithm is illustrated in Algorithm 13.

Algorithm 13 Dynamic Programming for the PWT problem

- Set $L = \max_{e_{ij} \in M} \mathcal{Z}(e_{ij})$, and $d_{in} = \sum_{l=i}^{n} d_l$, $1 \leq i \leq n$.
- Compute order $\preceq$ on the items e_{ij} by sorting them in lexicographic order with respect to their indices (i,j).
- For the first item e_{ij} according to $\preceq$, set $\zeta(i,j,0) = \mathcal{Z}(\emptyset)$ and $\zeta(i,j,w_{ij}) = \mathcal{Z}(e_{ij})$.
- Consider the remaining items of M in the order of $\preceq$ and do for each item e_{ij} and its predecessor $e_{i'j'}$:
  - In increasing order of k do for each $\zeta(i',j',k)$ with $\zeta(i',j',k) \neq -\infty$
    - If there is no $\zeta(i,j,k')$ with ($\zeta(i,j,k') \geq \zeta(i',j',k)$ and $k' < k$), set $\zeta(i,j,k) = max\{\zeta(i,j,k), \zeta(i',j',k)\}$.
    - If there is no $\zeta(i,j,k')$ with ($\zeta(i,j,k') \geq \zeta(i',j',k+w_{ij})$ and $k' < k+w_{ij}$), set $\zeta(i,j,k+w_{ij}) = max\{\zeta(i,j,k+w_{ij}), \zeta(i',j',k) + p_{ij} + Rd_{in}(\frac{1}{v_{\max}-\nu k} - \frac{1}{v_{max}-\nu(k+w_{ij})})\}$.

6.2 Fully Polynomial Time Approximation

We consider the amount that can be gained over the cost when the vehicle travels empty as the new objective $\mathcal{Z}'$ in order to avoid the situation that both positive and negative value of $\mathcal{Z}$ possibly exist. This is motivated by the scenario where the vehicle has to travel along the given route and the goal is to maximise the gain over this baseline cost. Note that an optimal solution for this objective is also an optimal solution for PWT. However, approximation results do not carry over to PWT as the objective values are "shifted" by the cost when travelling empty.

Let

$$\mathcal{Z}(\emptyset) = -R \cdot \sum_{i=1}^{n} d_i / v_{\max}$$

Algorithm 14 FPTAS for $\mathcal{Z}'(x)$

- Set $L = \max_{e_{ij} \in M} \mathcal{Z}'(e_{ij})$, $r = \epsilon L/m$, and $d_{in} = \sum_{l=i}^{n} d_l$, $1 \leq i \leq n$.
- Compute order $\preceq$ on the items e_{ij} by sorting them in lexicographic order with respect to their indices (i, j).
- For the first item e_{ij} according to $\preceq$, set $\zeta(i, j, 0) = \mathcal{Z}'(\emptyset)$ and $\zeta(i, j, w_{ij}) = \mathcal{Z}'(e_{ij})$.
- Consider the remaining items of M in the order of $\preceq$ and do for each item e_{ij} and its predecessor $e_{i'j'}$:
  - In increasing order of k do for each $\zeta(i', j', k)$ with $\zeta(i', j', k) \neq -\infty$
    - If there is no $\zeta(i, j, k')$ with ($\lfloor \zeta(i, j, k')/r \rfloor \geq \lfloor \zeta(i', j', k)/r \rfloor$ and $k' < k$), set $\zeta(i, j, k) = max\{\zeta(i, j, k), \zeta(i', j', k)\}$.
    - If there is no $\zeta(i, j, k')$ with ($\lfloor \zeta(i, j, k')/r \rfloor \geq \lfloor \zeta(i', j', k + w_{ij})/r \rfloor$ and $k' < k + w_{ij}$), set $\zeta(i, j, k + w_{ij}) = max\{\zeta(i, j, k + w_{ij}), \zeta(i', j', k) + p_{ij} + Rd_{in}(\frac{1}{v_{\max} - \nu k} - \frac{1}{v_{max} - \nu(k + w_{ij})})\}$.

be the travel cost (or reward) for the empty truck. $\mathcal{Z}(\emptyset)$ can be seen as the set up cost that we have to pay at least. We consider the objective

$$\mathcal{Z}'(\rho) = \mathcal{Z}(\rho) - \mathcal{Z}(\emptyset),$$

i. e. for the amount that we can gain over this setup cost, and give an FPTAS. Note, that we have $-R \cdot \mathcal{T}(\rho) \leq \mathcal{Z}(\emptyset)$ for any $\rho \in \{0, 1\}^m$ and $\mathcal{P}(\rho) - R \cdot \mathcal{T}(\rho) - \mathcal{Z}(\emptyset) = 0$ if $\rho = \{0\}^m$.

We now give a FPTAS for the amount that can be gained over the cost when the vehicle travels empty and denoted by OPT the optimal value for this objective, i.e.

$$OPT = \max_{\rho \in \{0,1\}^m} \mathcal{Z}'(\rho).$$

Considering the dynamic program for $\mathcal{Z}'(\rho)$ instead of $\mathcal{Z}(\rho)$ increases each entry by $|\mathcal{Z}(\emptyset)|$ and therefore obtains an optimal solution for $\mathcal{Z}'(\rho)$ in pseudo-polynomial time. In order to obtain an FPTAS, we round the values of $\mathcal{Z}'(\rho)$ and store for each rounded value only the minimal achievable weight.

Let

$$t(w) = \frac{1}{v_{\max} - \nu w}$$

denote the travel time per unit distance when travelling with weight w. Furthermore, let $w \geq 0$ be the weight of an item. We have $t(x + w) - t(x) \geq t(w) - t(0)$ for any $x \geq 0$ as t is a convex increasing function.

The pseudocode of the FPTAS is listed in Algorithm 14, from which we can state the following theorem.

Theorem 1. *Algorithm 14 is a fully polynomial time approximation scheme (FPTAS) for*

the objective $\mathcal{Z}'$. It obtains for any ϵ, $0 < \epsilon \leq 1$, a solution x with $\mathcal{Z}'(x) \geq (1-\epsilon) \cdot OPT$ in time $O(m^3/\epsilon)$.

The construction of the FPTAS only used the fact that the travel time per unit distance is monotonically increasing and convex. Hence, the FPTAS holds for any PWT problem where the travel time per unit distance has this property. The detailed proof of the theorem is listed in [129].

6.3 Experiments and Results

In this section, we investigate the effectiveness of the proposed DP and FPTAS approaches based on our implementations in Java. We mainly focus on two issues: 1) studying how the DP and FPTAS perform compared to the state-of-the-art approaches; 2) investigating how the performance and accuracy of the FPTAS change when the parameter ϵ is altered.

In order to be comparable to the mixed integer programming (MIP) and the branch-infer-and-bound (BIB) approaches presented in [140], we conduct our experiments on the same families of test instances. Our experiments are carried out on a computer with 4GB RAM and a 3.06GHz Intel Dual Core processor, which is also the same as the machine used in the paper mentioned above.

We compare the DP to the exact MIP (*exactMIP*) and the branch-infer-and-bound approaches as well as the FPTAS to the approximate MIP (*approxMIP*), as the former three are all exact approaches and the latter two are all approximations. Table 6.1 demonstrates the results for a route of 101 cities and various types of packing instances. For this particular family, we consider three types of instances: *uncorrelated* (uncorr), *uncorrelated with similar weights* (uncorr-s-w) and *bounded strongly correlated* (b-s-corr), which are further distinguished by the different correlations between profits and weights. In combination with three different numbers of items and three settings of the capacity, we have 27 instances in total, as shown in the column called "*Instance*". Similarly to the settings in [140], every instance with "_01" postfix has a relatively small capacity. We expect such instances to be potentially easy to solve by DP and FPTAS due to the nature of the algorithms. The *OPT* column shows the optimum of each instance and the *RT(s)* columns illustrate the running time for each of the approaches in the time unit of a second. To demonstrate the quality of an approximate approach applied to the instances, we use the ratio between the objective value obtained by the algorithm and the optimum obtained for an instance as the approximation rate $AR(\%) = 100 \times \frac{OZJ}{OPT}$.

In the comparison of exact approaches, our results show that the DP is much quicker than the exact MIP and BIB in solving the majority of the instances. The exact MIP is slower than the DP in every case and this dominance is mostly significant. For example, it spends around 35 minutes to solve the instance *uncorr-s-w_10* with $1,000$ items, where the DP needs around 15 seconds only. On the other hand, the BIB slightly beats the DP on three instances, but the DP is superior for the rest 24 instances. An extreme case is *b-s-corr_01* with $1,000$ items where the BIB spends above 1.5 hours while the DP solves it in 11 seconds only. Concerning the running time of the DP, it significantly increases only for the instances having large amount

of items with strongly correlated weights and profits, such as *b-s-corr_06* and *b-s-corr_10* with $1,000$ items. However, *b-s-corr_01* seems exceptional due to the limited capacity assigned to the instance.

Our comparison between the approximation approaches shows that the FPTAS has significant advantages as well. The approximation ratios remain 100% when ϵ equals 0.0001 and 0.01. Only when ϵ is set to 0.25, the FPTAS starts to output the results having similar accuracies as the ones of *approxMIP*. With regard to the performance, the FPTAS takes less running time than *approxMIP* on the majority of the instances despite the setting of ϵ. As an extreme case, *approxMIP* requires hours to solve the *uncorr-s-w_01* instance with $1,000$ items, but the FPTAS takes less than a second. However, the *approxMIP* performs much better on *b-s-corr_06* and *b-s-corr_10* with $1,000$ items. This somehow indicates that the underlying factors that make instances hard to solve by approximate MIP and FPTAS have different nature. Understanding these factors more and using them wisely should help to build a more powerful algorithm with mixed features of MIP and FPTAS.

In our second experiment, we use test instances which are slightly different to those in the benchmark used in [140]. This is motivated by our findings that relaxing ϵ from 0.0001 to 0.75 improves the runtime performance of FPTAS by around 50% for the b-s-corr instances, while it does not degrade the accuracy noticeably. At the same time, there is no significant improvement for other instances. It's surprising as it shows that the performance improvement can be easily achieved on complex instances. Therefore, we study how the FPTAS performs if the instances are more complicated. The idea is to use instances with large weights, which are known to be difficult regarding dynamic programming based approaches for the classical knapsack problem. We follow the same way to create TTP instances as proposed in [139] and generate the knapsack component of the problem as discussed in [138]. Specifically, we extend the range to generate potential profits and weights from $[1, 10^3]$ to $[1, 10^7]$ and focus on *uncorrelated* (uncorr), *uncorrelated with similar weights* (uncorr-s-w), and *multiple strongly correlated* (m-s-corr) types of instances. Additionally, in the stage of assigning the items of a knapsack instance to particular cities of a given TSP tour, we sort the items in descending order of their profits and the second city obtains k, $k \in \{1, 5, 10\}$, items of the largest profits, the third city then has the next k items, and so on. We expect that such assignment should force the algorithms to select items in the first cities of a route making the instances more challenging for the DP and FPTAS. In reality, these instances indeed are harder than the ones in the first experiment, which forces us to switch to the 128GB RAM and 8 $\times$ (2.8GHz AMD 6 core processors) cluster machine to carry out the second experiment.

Table 6.2 illustrates the results of running the DP and FPTAS on the instances with the large range of profits and weights. Generally speaking, we can observe that the instances are significantly harder to solve than those ones from the first experiment, that is they take comparably more time. Similarly, the instances with large number of items, larger capacity, and strong correlation between profits and weights are now hard for the DP as well. Oppositely to the results of the previous experiment, the FPTAS performs much better when dealing with such instances in the case when ϵ is relaxed. For example, its performance is improved by 95% for the instance *m-s-corr_10* with $1,000$ items when ϵ is raised from 0.0001 to 0.75 while the approximation rate remains at 100%.

TABLE 6.1: Results on Small Range Instances

Instance	m	OPT	Exact Approaches			Approximation Approaches											
			exactMIP	BIB	DP	approxMIP		FPTAS									
								$\epsilon = 0.0001$		$\epsilon = 0.01$		$\epsilon = 0.1$		$\epsilon = 0.25$		$\epsilon = 0.75$	
			RT(s)	RT(s)	RT(s)	AR(%)	RT(s)	AR(%)	RT(s)	AR(%)	RT(s)	AR(%)	RT(s)	AR(%)	RT(s)	AR(%)	RT(s)
instance family `eil101`																	
uncorr_01	100	1651.6970	1.217	5.694	0.027	100.0000	3.838	100.0000	0.001	100.0000	0.001	100.0000	0.001	100.0000	0.001	100.0000	0.025
uncorr_06	100	10155.4942	12.605	3.698	0.065	100.0000	4.961	100.0000	0.012	100.0000	0.011	100.0000	0.011	100.0000	0.011	99.9928	0.063
uncorr_10	100	10297.7134	3.525	0.795	0.036	100.0000	0.624	100.0000	0.017	100.0000	0.017	99.9939	0.016	99.9939	0.016	99.9653	0.037
uncorr-s-w_01	100	2152.6188	0.328	7.566	0.001	100.0000	3.978	100.0000	0.000	100.0000	0.000	100.0000	0.000	100.0000	0.000	100.0000	0.003
uncorr-s-w_06	100	4333.8512	12.590	2.215	0.012	100.0000	2.699	100.0000	0.008	100.0000	0.007	100.0000	0.007	99.9569	0.008	99.9569	0.017
uncorr-s-w_10	100	9048.4908	37.144	1.107	0.022	100.0000	1.763	100.0000	0.012	100.0000	0.012	100.0000	0.012	100.0000	0.013	99.9355	0.020
b-s-corr_01	100	4441.9852	1.420	125.954	0.014	100.0000	5.366	100.0000	0.010	100.0000	0.009	100.0000	0.009	100.0000	0.008	100.0000	0.013
b-s-corr_06	100	10260.9767	4.509	22.541	0.101	100.0000	2.761	100.0000	0.058	100.0000	0.057	100.0000	0.048	100.0000	0.043	100.0000	0.087
b-s-corr_10	100	13630.6153	11.013	27.081	0.187	99.9971	3.713	100.0000	0.103	100.0000	0.101	99.9971	0.081	99.9606	0.065	99.8143	0.113
uncorr_01	500	17608.5781	19.594	27.581	0.247	100.0000	5.757	100.0000	0.171	100.0000	0.161	100.0000	0.153	100.0000	0.163	100.0000	0.377
uncorr_06	500	56294.5239	384.213	13.354	2.829	100.0000	7.800	100.0000	2.370	100.0000	2.344	100.0000	2.300	100.0000	2.212	100.0000	2.340
uncorr_10	500	66141.4840	211.302	2.325	4.010	100.0000	0.718	100.0000	3.720	100.0000	3.645	100.0000	3.446	100.0000	3.531	100.0000	3.632
uncorr-s-w_01	500	13418.8406	4.337	34.866	0.090	100.0000	50.310	100.0000	0.085	100.0000	0.090	100.0000	0.084	100.0000	0.087	99.9910	0.085
uncorr-s-w_06	500	34280.4730	346.430	7.285	1.040	100.0000	9.609	100.0000	0.964	100.0000	0.933	100.0000	0.905	100.0000	0.936	100.0000	0.920
uncorr-s-w_10	500	50836.6588	519.902	3.338	2.022	100.0000	3.354	100.0000	2.005	100.0000	1.783	100.0000	1.753	100.0000	1.784	100.0000	2.147
b-s-corr_01	500	21306.9158	40.482	624.204	1.534	100.0000	13.338	100.0000	1.373	100.0000	1.279	100.0000	1.116	100.0000	0.949	100.0000	0.716
b-s-corr_06	500	69370.2367	236.387	97.313	14.616	99.9996	7.847	100.0000	13.393	100.0000	12.975	100.0000	11.642	99.9996	9.741	99.9996	6.018
b-s-corr_10	500	82033.9452	376.569	218.728	22.011	100.0000	2.309	100.0000	21.372	100.0000	20.829	100.0000	18.573	100.0000	15.313	99.9943	8.840
uncorr_01	1000	36170.9109	218.306	114.567	1.872	99.9993	11.918	100.0000	1.891	100.0000	1.875	100.0000	1.832	100.0000	1.845	100.0000	1.764
uncorr_06	1000	93949.1981	1261.949	36.847	20.944	100.0000	17.971	100.0000	17.024	100.0000	16.615	100.0000	16.545	100.0000	16.378	100.0000	15.713
uncorr_10	1000	122963.6617	620.896	4.821	30.116	100.0000	2.184	100.0000	27.305	100.0000	26.783	100.0000	26.541	100.0000	26.051	100.0000	23.905
uncorr-s-w_01	1000	27800.9614	241.957	399.158	0.802	100.0000	4985.566	100.0000	0.730	100.0000	0.690	100.0000	0.688	100.0000	0.724	100.0000	0.687
uncorr-s-w_06	1000	61764.4599	1152.624	12.792	9.872	100.0000	19.063	100.0000	8.686	100.0000	8.812	100.0000	8.560	100.0000	8.740	100.0000	8.396
uncorr-s-w_10	1000	103572.4074	2146.408	7.644	15.047	100.0000	9.688	100.0000	14.030	100.0000	13.912	100.0000	13.797	100.0000	13.982	100.0000	13.492
b-s-corr_01	1000	46886.1094	378.551	6129.531	11.783	99.9988	46.394	100.0000	11.714	100.0000	11.358	100.0000	10.793	100.0000	9.592	100.0000	6.536
b-s-corr_06	1000	125830.6887	643.533	919.201	94.523	99.9999	10.311	100.0000	92.411	100.0000	91.039	100.0000	83.002	99.9999	71.078	100.0000	45.433
b-s-corr_10	1000	161990.5015	862.572	1646.520	151.601	100.0000	7.160	100.0000	150.279	100.0000	149.722	100.0000	134.764	100.0000	113.049	99.9981	70.135

TABLE 6.2: Results of DP and FPTAS on Large Range Instances

Instance	m	DP		FPTAS											
				$\epsilon = 0.0001$		$\epsilon = 0.001$		$\epsilon = 0.01$		$\epsilon = 0.1$		$\epsilon = 0.25$		$\epsilon = 0.5$		$\epsilon = 0.75$	
		OPT	RT(s)	AR(%)	RT(s)	AR(%)	RT(s)	AR(%)	RT(s)	AR(%)	RT(s)	AR(%)	RT(s)	AR(%)	RT(s)	AR(%)	RT(s)
instance family `eil101_large-range`																	
uncorr_01	100	69802802.2801	0.030	100.0000	0.002	100.0000	0.002	100.0000	0.002	100.0000	0.002	100.0000	0.002	100.0000	0.002	100.0000	0.029
uncorr_06	100	204813765.6933	0.053	100.0000	0.019	100.0000	0.020	100.0000	0.019	100.0000	0.019	100.0000	0.019	100.0000	0.019	100.0000	0.049
uncorr_10	100	17217618 2.1249	0.041	100.0000	0.028	100.0000	0.028	100.0000	0.028	100.0000	0.028	100.0000	0.027	100.0000	0.026	99.9628	0.037
uncorr-s-w_01	100	36420530.5753	0.006	100.0000	0.003	100.0000	0.003	100.0000	0.003	100.0000	0.003	100.0000	0.003	100.0000	0.002	100.0000	0.004
uncorr-s-w_06	100	148058928.2952	0.098	100.0000	0.072	100.0000	0.502	100.0000	0.072	100.0000	0.069	100.0000	0.065	100.0000	0.059	100.0000	0.070
uncorr-s-w_10	100	142538516.4602	0.136	100.0000	0.101	100.0000	0.104	100.0000	0.103	99.9978	0.096	99.9978	0.086	99.9978	0.073	99.9978	0.089
m-s-corr_01	100	19549602.2671	0.003	100.0000	0.002	100.0000	0.002	100.0000	0.002	100.0000	0.002	100.0000	0.002	100.0000	0.001	100.0000	0.002
m-s-corr_06	100	137203175.1921	0.147	100.0000	0.115	100.0000	0.118	100.0000	0.113	100.0000	0.089	100.0000	0.063	100.0000	0.040	100.0000	0.043
m-s-corr_10	100	225584278.6004	0.424	100.0000	0.326	100.0000	0.329	100.0000	0.312	100.0000	0.200	100.0000	0.179	100.0000	0.086	100.0000	0.073
uncorr_01	500	385692662.0930	0.470	100.0000	0.451	100.0000	0.454	100.0000	0.619	100.0000	0.508	100.0000	0.445	100.0000	0.430	100.0000	0.517
uncorr_06	500	958013934.6172	3.539	100.0000	3.749	100.0000	7.431	100.0000	3.947	100.0000	3.690	99.9996	3.677	99.9996	3.486	99.9993	3.021
uncorr_10	500	844949838.4389	4.870	100.0000	5.393	100.0000	5.716	100.0000	5.483	100.0000	5.135	100.0000	4.851	99.9992	4.609	99.9992	4.295
uncorr-s-w_01	500	182418888.9364	1.157	100.0000	1.157	100.0000	1.199	100.0000	1.145	99.9995	1.112	99.9995	1.063	99.9995	0.977	99.9904	0.929
uncorr-s-w_06	500	780432253.0187	22.390	100.0000	25.040	100.0000	26.276	100.0000	24.024	100.0000	23.282	99.9997	21.756	99.9997	18.293	99.9997	18.411
uncorr-s-w_10	500	714433353.7957	30.959	100.0000	34.458	100.0000	39.004	100.0000	34.308	100.0000	32.308	99.9996	28.792	99.9990	26.392	99.9990	25.971
m-s-corr_01	500	96463941.1275	2.335	100.0000	2.478	100.0000	2.782	100.0000	2.695	100.0000	1.509	100.0000	0.963	100.0000	0.546	100.0000	0.408
m-s-corr_06	500	666701000.1488	108.705	100.0000	126.833	100.0000	139.630	100.0000	122.750	100.0000	62.479	100.0000	33.547	100.0000	17.959	100.0000	10.642
m-s-corr_10	500	1082009880.5886	262.999	100.0000	299.862	100.0000	317.352	100.0000	274.284	100.0000	145.087	100.0000	78.470	99.9994	41.816	99.9994	25.924
uncorr_01	1000	77738636.9660	4.222	100.0000	4.397	100.0000	4.347	100.0000	4.309	100.0000	4.341	100.0000	4.377	100.0000	4.280	100.0000	4.240
uncorr_06	1000	1933319297.4248	46.043	100.0000	51.383	100.0000	53.087	100.0000	48.861	100.0000	52.957	99.9999	52.062	99.9997	50.286	99.9996	51.488
uncorr_10	1000	1693797490.1704	64.485	100.0000	76.744	100.0000	78.847	100.0000	74.128	100.0000	82.754	100.0000	77.057	100.0000	72.283	100.0000	72.567
uncorr-s-w_01	1000	361991311.8336	14.254	100.0000	15.072	100.0000	15.670	100.0000	14.523	100.0000	14.110	100.0000	14.039	100.0000	12.088	100.0000	11.129
uncorr-s-w_06	1000	1574469459.3163	286.843	100.0000	318.096	100.0000	330.508	100.0000	337.289	100.0000	334.318	100.0000	307.588	99.9998	270.013	99.9996	245.927
uncorr-s-w_10	1000	1439410696.3695	393.793	100.0000	438.775	100.0000	455.830	100.0000	464.527	100.0000	441.955	100.0000	433.672	99.9994	378.917	99.9994	340.813
m-s-corr_01	1000	191170309.5684	46.858	100.0000	58.031	100.0000	59.987	100.0000	58.101	100.0000	31.703	100.0000	18.771	100.0000	10.728	100.0000	6.831
m-s-corr_06	1000	1315708161.7720	2393.205	100.0000	2512.281	100.0000	2606.412	100.0000	1921.573	100.0000	666.749	100.0000	364.452	100.0000	208.969	100.0000	150.060
m-s-corr_10	1000	2163713055.3759	6761.490	100.0000	6668.535	100.0000	6441.906	100.0000	4526.653	100.0000	1334.882	100.0000	703.258	100.0000	397.527	100.0000	282.211

6.4 Conclusion

This and the previous chapter cover our studies of the packing while travelling (PWT) problem, which is a simplified travelling thief problem (TTP) that focus on the nonlinearity of transportation costs in the TTP. In this chapter, we designed a dynamic programming algorithm that solves the problem in pseudo-polynomial time. Furthermore, we have shown that the original objective of the problem is hard to approximate and have given an FPTAS for optimising the amount that can be gained over the smallest possible travel cost. It should be noted that the FPTAS applies to a wider range of problems as our proof only assumes that the travel cost per unit of distance in terms of weight w is monotone increasing and convex. Our experimental results on different types of knapsack instances show the advantage of the dynamic programming over the previous approach, based on mixed integer programming and branch-infer-and-bound concepts. Furthermore, we have demonstrated the effectiveness of the FPTAS on instances with a large weight and profit range. From the next chapter onwards, we will start to focus on the TTP, especially on extending our investigations on the PWT back to the TTP.

Chapter 7

Exact Approaches for the Travelling Thief Problem

We have learned in the previous chapter (Chapter 6) that the packing while travelling (PWT, in Section 2.3.4) problem, the simplified version of the TTP, can be solved via dynamic programming (DP), taking into account the fact that the weights are integers. By following the traditional scheme and sequentially examining every possible weights instead of every possible combination of items to the point of exhaustion, the DP can solve the PWT problem within pseudo-polynomial time. In this chapter, we extend our findings from the PWT problem, in order to form the exact approaches needed for the TTP.

Our first algorithm is a DP for the TTP. By combining the DP for the PWT problem and the classical Held-Karp algorithm for the TSP [72] organically, we have our first approach. Even though the algorithm improves the time complexity of solving the TTP from brute-force to $\mathcal{O}(2^n n\,(n + mC))$, its memory consumption is still expensive and this ultimately becomes a major obstruction. We therefore introduce an upper bound on the value of a feasible solution that can be derived from the partial solutions, which then significantly reduces the search space. Furthermore, a branch and bound search (BnB), taking the advantage of the upper bound, and a constraint programming (CP) adopting the existing state-of-the-art paradigm are introduced for comparison with the DP method.

As introduced in Section 3.5, many approximate approaches have been introduced for addressing the TTP. Most of them are either evolutionary or heuristic. Although such approximate approaches seem to perform well in the majority of the instances in the benchmark library of TTP Polyakovskiy et al. [139], none of the methods can be evaluated with respect to their accuracy, even for small TTP instances, due to the lack of exact methods or known upper bounds. Our exact techniques address this issue well on the small size instances that we propose, based on the reduction of the existing problem instances in the benchmark library. Our approaches thus help to build a more comprehensive review of the approximate approaches.

This chapter is based on a conference paper published at the Simulated Evolution and Learning International Conference (SEAL) [176]. In the remainder of this chapter, we introduce our exact approaches from Section 7.1. In Section 7.2, we elaborate on the setup of our experiments and compare our exact and hybrid approaches with the best approximate ones, which forms an incomplete overview of the accuracy of the state-of-the-art approximate approaches. The conclusions are drawn in Section 7.3.

7.1 Exact Approaches to the TTP

This section describes the DP and then continues with a branch and bound approach (BnB) and with a constraint programming (CP) technique adopted for the TTP.

The DP algorithm introduced in our previous chapter (Chapter 6) can solve the PWT problem in pseudo-polynomial time by considering the fact that the weights are integer. This means that the DP maps every possible weight $w \in [0, C]$ to a packing plan ρ, i.e. $f : w \mapsto \rho$, which guarantees a certain profit. Then the optimal packing plan ρ^* is to be selected among all the plans that have been obtained. Herein, two of our exact approaches are based on these findings. For the consistency in the following paragraphs, we let $\mathcal{Z}(\pi, \rho)$ denote the objective function of TTP with respect to a solution (π, ρ), π and ρ are the tour and the packing plan respectively, as defined in Section 2.3. And let $\cdot$ denote all possible weights for a given TTP instance.

7.1.1 Dynamic Programming

Our DP is based on the Held-Karp algorithm for the TSP [72] augmented by the dynamic programming routine [129] applied to resolve low level PWT subproblems. We consider a subset of cities $S \subseteq N$ and a city $j \in S$. Let $A(S, j, w)$ be the maximum reward of the path visiting each city in S exactly once, starting at the home city and ending at j with the total knapsack's weight of w. Please notice that we exclude the travel from the last city back to the start city in this setting, which means if $|S| > 1$, we let $A(S, 1, w) = -\infty$ for any $w \in [0, C]$.

Our base case consists of $A(\{1\}, 1, 0) = 0$ and of $A(\{1\}, 1, w) = -\infty$ for $0 < w \leq C$. Our general case for $A(S, j, w)$ is based on $A\left(S \setminus \{j\}, i, w - \overline{W}_j\right)$, which is the path from city 1 to city $i \in S$, plus the reward gained from visiting j right after i. In fact, i must be the best choice:

$$A(S, j, w) = \max_{i \in S : i \neq j} \left\{ A\left(S \setminus \{j\}, i, w - \overline{W}_j\left(S \setminus \{j\}, i\right)\right) + \overline{P}_j\left(S \setminus \{j\}, i\right) - \frac{Rd_{ij}}{v_{max} - \nu w} \right\}.$$

Here, $\overline{W}_j\left(S \setminus \{j\}, i\right)$ and $\overline{P}_j\left(S \setminus \{j\}, i\right)$ represent the total weight and the total profit of the items chosen in city j. They both result from the best solution of the PWT subproblem, where a subset of items in M_j must be optimally chosen with respect to the set of partial solutions corresponding to $A(S \setminus \{j\}, i, w)$, $w \in [0, C]$. In fact, the sub-problem considers only the items of city j and can be solved via the dynamic programming approach for the PWT [129].

We start computing solutions with all subsets of size $s = 2$ and calculate $A(S, j, w)$ sequentially for all possible knapsack's weights w and subsets $S \subseteq N$ subject to S containing city 1. We iteratively increment s and continue until $s = n$. Finally, we compute the value of an optimal solution for the complete tour as

$$\max_{i \in S : i \neq 1} \left\{ A(N, i, w) - \frac{Rd_{i1}}{v_{max} - \nu w} \right\}.$$

The pseudocode of our DP to the TTP is listed in Algorithm 15.

There are at most $2^n n$ subproblems, and each of them takes the time of $\mathcal{O}(n+mC)$. Therefore, the total running time is $\mathcal{O}(2^n n\,(n + mC))$. The dynamic programming is

Algorithm 15 Dynamic programming to the TTP

set $A(\{1\}, 1, 0) = 0$
for $w = 1$ to C **do**
    set $A(\{1\}, 1, w) = -\infty$
end for
for $s = 2$ to n **do**
    for any $S \subseteq N : |S| = s,\ 1 \in S$ **do**
        for $w = 0$ to C **do**
            set $A(S, 1, w) = -\infty$
            for any $j \in S,\ j \neq 1$ **do**
                compute $A(S, j, w) =$
                $\max_{i \in S: i \neq j} \left\{ A\left(S \setminus \{j\}, i, w - \overline{W_j}(S \setminus \{j\}, i)\right) + \overline{P_j}(S \setminus \{j\}, i) - \frac{Rd_{ij}}{v_{max} - \nu w} \right\}$
            end for
        end for
    end for
end for
return $\max_{i \in S: i \neq 1} \left\{ A(N, i, w) - \frac{Rd_{i1}}{v_{max} - \nu w} \right\}$

also rather expensive in terms of the memory consumption, which reaches $\mathcal{O}(2^n nC)$. In order to speed up computations, let us define an upper bound on the value of a feasible solution that can be derived from the partial solutions corresponding to $A(S \setminus \{j\}, i, w)$, for any $w \in [0, C]$, as follows:

$$E_U(A(S, j, \cdot)) = \max_{0 \leq w \leq W} A(S, j, w) + \sum_{k \in N \setminus S} \sum_{l=1}^{m_k} p_{kl} - \frac{Rd_{j1}}{v_{max}}$$

It estimates the maximal profit that the thief may obtain by passing the remaining part of the tour with the maximal speed; that is, generating the minimal possible cost of travelling. Obviously, an optimal solution must not exceed this bound. Therefore, if an incumbent solution $\mathcal{Z}(\pi', \rho')$ exists, those partial solutions whose $E_U(A(S, j, w)) < \mathcal{Z}(\pi', \rho')$ can be ignored. In practice, we can obtain an incumbent solution in two stages. First, a feasible solution π' for the TSP part of the problem can be computed by a TSP solver such as Concorde [4] or by the Lin-Kernighan algorithm [101]. Second, the dynamic programming for PWT can be applied to determine the best packing plan ρ' for π'.

7.1.2 Branch and Bound Search

Now, we introduce a branch and bound search for the TTP employing the upper bound E_U defined in Section 7.1.1, which is adapted from the generic Branch and Bound Search introduced in Section 3.2.2. Algorithm 16 depicts the pseudocode, where $\pi_i, i \in \{1, ..., n\}$ denotes a sub-permutation of π with the cities 1 to i visited, and f_i is the mapping $f : w \mapsto \rho$ calculated for π_i by the dynamic programming for the PWT.

Algorithm 16 Branch and Bound Search for the TTP

```
1: procedure BNB SEARCH
2:     Create an initial solution to gain the benefit best and an tour permutation π
3:     Create an empty mapping M
4:     Set l = 0
5:     SEARCH(π, l, M, best)
6: end procedure
7: function SEARCH(π, l, M, best)
8:     if l == n then
9:         calculate Z(π, f_n(·)) from Z(π_{n−1}, f_{n−1}(·)) in M
10:        return max{max Z(π, f_n(·)), best}
11:    else
12:        for i = l + 1 to n do
13:            Swap cities l + 1 and i in π
14:            Set M' = Calculate Z(π_{l+1}, f_{l+1}(·)) from Z(π_l, f_l(·)) in M
15:            if max E_U(π_{l+1}, f_{l+1}(·)) > best then
16:                best = max{best, SEARCH(π, l + 1, M', best) }
17:            end if
18:            Swap cities l + 1 and i in π
19:        end for
20:        return best
21:    end if
22: end function
```

A way to tighten the upper bound E_U is by providing a better estimation of the remaining distance from the current city k to the last city of the tour. Currently, the shortest distance from k to 1, i.e. d_{k1}, is used. The following two ways can improve the estimation: (i) the use of distance d_{f1} from city f to city 1, where f is the farthest unvisited city from 1; (ii) the use of distance $d^* - d_t$, where d^* is the shortest path that can be pre-calculated and d_t is the distance passed so far to achieve city k in the tour π. These two ideas can be joined together by using the $\max\{d_f, (d^* - d_t)\}$ to enhance the result.

7.1.3 Constraint Programming

Now, we present our third exact approach adopting the existing state-of-the-art constraint programming (CP) paradigm [78]. Our model employs a simple permutation based representation of the tour which allows the use of the `AllDifferent` filtering algorithm [15]. Similarly to the Section 2.3, a vector $W = (W_1, \ldots, W_n)$ is used to refer to the total weights accumulated in the cities of tour π. Specifically, W_i is the weight of the knapsack when the thief departs from city i. The model bases the search on two types of decision variables:

- $\boldsymbol{x}$ denotes the particular positions of the cities in tour π. Variable x_i takes the value of $j \in N$ to indicate that j is the ith city to be visited. The initial variable domain of x_1 is $D(x_1) = \{1\}$ and it is $D(x_i) = N \setminus \{i\}$ for any subsequently visited city $i = 2, \ldots, n$.

$$max \sum_{i=1}^{n} \sum_{j=1}^{m_i} p_{ij} y_{ij}$$

$$- R\left(\sum_{i=1}^{n-1} \frac{\texttt{Element}(d, n\,(x_i - 1) + x_{i+1})}{v_{max} - \nu \texttt{Element}(W, x_i)} + \frac{\texttt{Element}(d, n\,(x_n - 1) + 1)}{v_{max} - \nu \texttt{Element}(W, x_n)} \right) \quad (7.1)$$

$$\texttt{AllDifferent}[x_1, \ldots, x_n] \quad (7.2)$$

$$W_i = W_{i-1} + \sum_{j \in M_i} w_{ij} y_{ij}, \;\; i \in \{2, \ldots, n\} \quad (7.3)$$

$$W_n \leq C \quad (7.4)$$

FIGURE 7.1: Constraint programming model to the TTP

- $\boldsymbol{y}$ signals on the selection of an item in the packing plan ρ. Variable y_{ik}, $i \in N$, $k \in M_i$, is binary, therefore $D\,(y_{ik}) = \{0, 1\}$.

Furthermore, an integer-valued vector d is used to express the distance matrix so that its element $n\,(x_i - 1) + x_{i+1}$ equals the distance $d_{x_i x_{i+1}}$ between two consecutive cities x_i and x_{i+1} in π.

The model relies on the $\texttt{AllDifferent}[x_1, \ldots, x_n]$ constraint, which ensures that the values of $x_1, \ldots, x_n$ are distinct. It also involves the $\texttt{Element}(g, h)$ expression, which returns the hth variable in the list of variables g. In total, the model (CPTTP) consists of the objective function and constraints as depicted in the Figure (7.1). Expression (7.1) calculates the objective value according to function (2.11). Constraint (7.2) verifies that all the cities are assigned to different positions, and thus are visited exactly once. This is a sub-tour elimination constraint. Equation (7.3) calculates the weight W_i of all the items collected in the cities $1, \ldots, i$. Equation (7.4) is a capacity constraint.

The performance of a CP model depends on its solver; specifically, on the filtering algorithms and on the search strategies it applies. Here, we use IBM ILOG CP OPTIMISER 12.6.2 with its searching algorithm set to the *restart mode*. This mode adopts a general purpose search strategy [143] inspired from integer programming techniques and is based on the concept of the impact of a variable. The impact measures the importance of a variable in reducing the search space. The impacts, which are learned from the observation of the domains' reduction during the search, help the restart mode dramatically improve the performance of the search. Within the search, the cities are assigned to the positions first and then the items are decided on. Therefore, the solver instantiates $x_1, \ldots, x_n$ prior to $y_{21}, \ldots, y_{nm_n}$ variables applying its default selection strategy. Our extensive study shows that such an order gives the best results fast.

7.2 Computational Experiments

In this section, we first compare the performance of the exact approaches to TTP in order to find the best one for setting the baseline for the subsequent comparison

of the approximate approaches. Our experiments run on the CPU cluster of the Phoenix HPC at the University of Adelaide, which contains 3072 Intel(R) Xeon(R) 2.30GHz CPU cores and 12TB of memory. We allocate one CPU core and 32GB of memory to each individual experiment.

7.2.1 Computational Set Up

Instance	n	m	Running time (in seconds) DP	BnB	CP
eil51_n05_m4_uncorr_01	5	4	0.018	0.023	0.222
eil51_n06_m5_uncorr_01	6	5	0.07	0.079	0.24
eil51_n07_m6_uncorr_01	7	6	0.143	0.195	0.497
eil51_n08_m7_uncorr_01	8	7	0.343	0.505	4.594
eil51_n09_m8_uncorr_01	9	8	0.633	1.492	63.838
eil51_n10_m9_uncorr_01	10	9	0.933	5.188	776.55
eil51_n11_m10_uncorr_01	11	10	2.414	23.106	12861.181
eil51_n12_m11_uncorr_01	12	11	3.938	204.786	-
eil51_n13_m12_uncorr_01	13	12	14.217	2007.074	-
eil51_n14_m13_uncorr_01	14	13	13.408	36944.146	-
eil51_n15_m14_uncorr_01	15	14	89.461	-	-
eil51_n16_m15_uncorr_01	16	15	59.526	-	-
eil51_n17_m16_uncorr_01	17	16	134.905	-	-
eil51_n18_m17_uncorr_01	18	17	366.082	-	-
eil51_n19_m18_uncorr_01	19	18	830.18	-	-
eil51_n20_m19_uncorr_01	20	19	2456.873	-	-

TABLE 7.1: Columns 'n' and 'm' denote the number of cities and the number of items, respectively. Running times are given in seconds for DP, BnB and CP for different numbers of cities and items. '-' denotes the case when an approach failed to achieve an optimal solution in the given time limit.

To run our experiments, we generate an additional set of small-sized instances following the way proposed in [139][1]. We use only a single instance of the original TSP library [145] as the starting point for our new subset. It is entitled as `eil51` and contains 51 cities. Out of these cities, we select uniformly at random cities that we removed in order to obtain smaller test problems with $n = 5, \ldots, 20$ cities. To set up the knapsack component of the problem, we adopt the approach given in [138] and use the corresponding problem generator available in [137]. As one of the input parameters, the generator asks for the range of coefficients, which we set to 1000. In total, we create knapsack test problems containing $k(n-1)$, $k \in \{1, 5, 10\}$ items and which are characterised by a knapsack capacity category $Q \in \{1, 6, 10\}$. Our experiments focus on *uncorrelated* (`uncorr`), *uncorrelated with similar weights* (`uncorr-s-w`), and *multiple strongly correlated* (`m-s-corr`) types of instances. At

[1] All instances are available online:
http://cs.adelaide.edu.au/~optlog/research/ttp.php

the stage of assigning the items of a knapsack instance to the particular cities of a given TSP tour, we sort the items in descending order of their profits and the second city obtains k, $k \in \{1, 5, 10\}$, items of the largest profits, the third city then has the next k items, and so on. All the instances use the "CEIL_2D" for intra-city distances, which means that the Euclidean distances are rounded up to the nearest integer. We set v_{min} and v_{max} to 0.1 and 1.

Tables 7.1 and 7.3 illustrate the results of the experiments. The test instances' names should be read as follows. First, `eil51` stays for the name of the original TSP problem. The values succeeding n and m denote the actual number of cities and the total number of items, respectively, which are further followed by the generation type of a knapsack problem. Finally, the postfixes 1, 6 and, 10 in the instances' names describe the knapsack's capacity C.

7.2.2 Comparison of the exact approaches

We compare the three exact algorithms by allocating each instance a generous 24-hour time limit. Our aim is to analyse the running time of the approaches influenced by the increasing number of cities. Table 7.1 shows the running time of the approaches, which illustrates the leading performance of the DP among the three algorithms. The DP managed to solve the instance with 20 cities in less than one hour, while BnB and CP can only handle the instances with 13 and 10 cities respectively in such time frame.

7.2.3 Comparison between DP and Approximate Approaches

With the exact approaches being introduced, approximate approaches can be evaluated with respect to their accuracy to the optima. In the case of the TTP, most state-of-the-art approximate approaches are evolutionary algorithms and local searches, such as Memetic Algorithm with 2-OPT and Bit-flip (MA2B), CoSolver-based with 2-OPT, and Simulated Annealing (CS2SA) in [50], CoSolver-based with 2-OPT and Bit-flip (CS2B) in [49], and S1, S5, and C5 in [56].

In addition to existing heuristics, we introduce enhanced approaches of S1 and S5, which are hybrids of the two and that one of dynamic programming for the PWT [129]. The original S1 and S5 work as follows. First, a single TSP tour is computed using the Chained Lin-Kernighan-Heuristic [5], then a fast packing heuristic is applied. S1 performs these two steps only once and only in this order, while S5 repeats S1 until the time budget is exhausted. Our hybrids DP-S1 and DP-S5 are equivalent to these two algorithms, however, they use the exact dynamic programming to the PWT as a packing solver. This provides better results as we can now compute the optimal packing for the sampled TSP tours.

We start by showing a performance summary of 10 algorithms on 432 instances in Table 7.2. In addition, Table 7.3 shows detailed results for a subset of the best approaches on a subset of instances. Figure 7.2 shows the results of the entire comparison. We include trend lines[2] for two selected approaches, which we will explain in the following.

We would like to highlight the following observations:

[2]They are fitted polynomials of degree six used only for visualisation purposes.

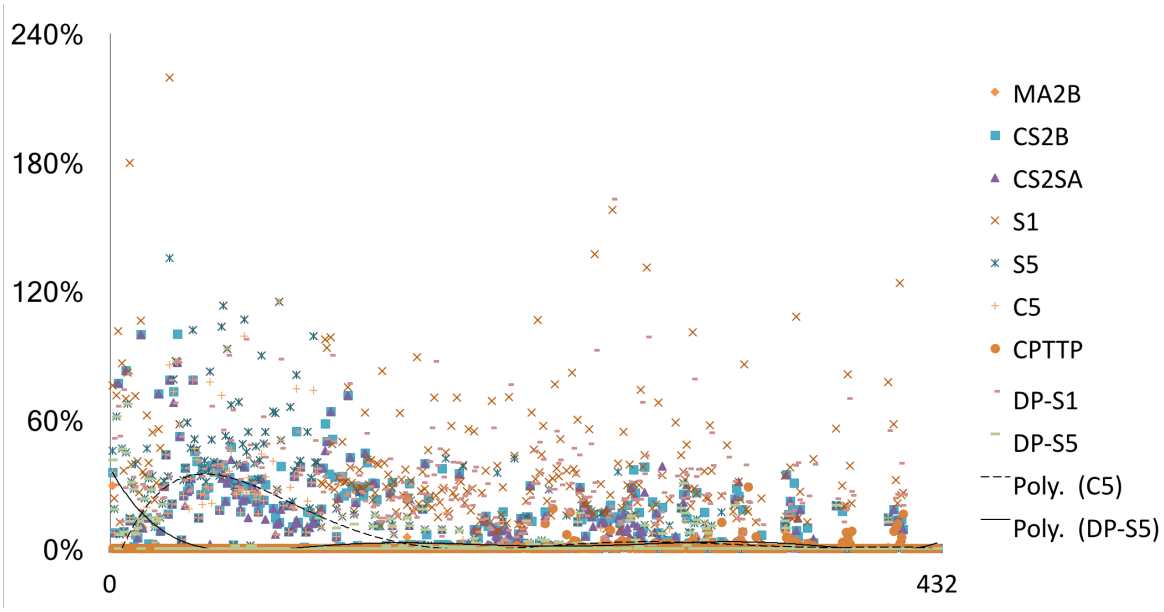


FIGURE 7.2: Showing a gap to an optimal solution when one has been obtained by an exact approach. From left to right: the 432 instances first sorted by the number of cities, then by the total number of items.

gap	MA2B	CS2B	CS2SA	S1	S5	C5	DP-S1	DP-S5
avg	0.3%	15.3%	11.5%	38.9%	15.7%	09.9%	30.1%	3.3%
stdev	2.2%	17.8%	16.7%	29.4%	24.6%	18.8%	20.1%	8.5%
$\#_{opt}$	312	70	117	3	42	193	5	85
$\#_{1\%}$	265	100	132	10	160	193	9	245
$\#_{10\%}$	324	161	206	27	203	240	33	288

TABLE 7.2: Performance summary of heuristic TTP solvers across all instances for which the optimal result has been obtained. $\#_{opt}$ is the number of times when the average of 10 independent repetitions is equal to the optimum. $\#_{1\%}$ and $\#_{10\%}$ show the number of times the averages are within 1% and 10%.

1. S1 performs badly across a wide range of instances. Its restart variant S5 performs better, however, its lack of a local search becomes apart in its relatively bad performance (compared to other approaches) on small instances.

2. C5 performs better than both S1 and S5, which is most likely due to its local searches that differentiate it from S1 and S5. Still, we can see a "hump" in its trend line for smaller instances, which flattens out quickly for larger instances.

3. The dynamic programming variants DP-S1 and DP-S5 perform slightly better than S1 and S5, which shows the difference in quality of the packing strategy; however, this is at times balanced out by the faster packing which allows more TSP tours to be sampled. For small instances, DP-S5 lacks a local search on the tours, which is why its gap to the optimum is relatively large, as shown by the respective trend lines.

4. MA2B dominates the field with outstanding performance across all instances, independent of number of cities and number of items. What is remarkable is the high reliability with which it reaches a global optimum.

Interestingly, all approaches seem to have difficulties solving instances with the knapsack configuration multiple-strongly-corr_01 (see Table 7.3). Compared to the other two knapsack types, TTP-DP takes the longest to solve the strongly correlated ones. Also, these tend to be the only instances for which the heuristics rarely find optimal solutions, if at all.

7.3 Conclusion

In this chapter, we extended our exact dynamic programming for the packing while travelling problem back to the travelling thief problem (TTP). We have presented and evaluated three exact approaches for the TTP based on dynamic programming, branch and bound, and constraint programming. The dynamic programming of the TTP, the best of the three, managed to achieve the optimal solutions for 432 instances, which facilitates further researches on evaluating the performance of current state-of-the-art TTP solvers. Our empirical investigations show that they are obtaining, in most cases, close to optimal solutions. However, for a small fraction of tested instances we observe a gap in the optimal solution of more than 10%. In the next chapter, we will extend our study to a multi-objective version of the TTP.

	TTP-DP		MA2B			C5		DP-S5	
Instance	OPT	RT	Gap	Std	RT	Gap	Std	Gap	Std
eil51_n05_m4_multiple-strongly-corr_01	619.227	0.02	29.1	12.1	2.71	35.5	1.20e-6	41.3	0.0
eil51_n05_m4_uncorr_01	466.929	0.02	0.0	0.0	3.22	0.0	2.20e-6	0.0	2.20e-6
eil51_n05_m4_uncorr-similar-weights_01	299.281	0.02	0.0	0.0	3.21	7.8	2.40e-6	7.8	1.20e-6
eil51_n05_m20_multiple-strongly-corr_01	773.573	0.08	13.4	0.0	1.44	14.3	0.0	12.8	0.0
eil51_n05_m20_uncorr_01	2144.796	0.07	0.0	0.0	3.35	7.4	0.0	6.6	2.30e-6
eil51_n05_m20_uncorr-similar-weights_01	269.015	0.04	0.0	0.0	3.51	0.0	2.30e-6	0.0	0.0
eil51_n10_m9_multiple-strongly-corr_01	573.897	1.21	0.0	0.0	6.07	0.0	0.0	0.0	0.0
eil51_n10_m9_uncorr_01	1125.715	0.93	0.0	0.0	6.06	0.0	1.30e-6	0.0	1.30e-6
eil51_n10_m9_uncorr-similar-weights_01	753.230	0.86	0.0	0.0	5.87	0.0	0.0	0.0	0.0
eil51_n10_m45_multiple-strongly-corr_01	1091.127	14.89	0.0	0.0	7.99	0.0	0.0	0.0	0.0
eil51_n10_m45_uncorr_01	6009.431	6.39	0.0	0.0	8.6	6.6	2.30e-6	0.0	0.0
eil51_n10_m45_uncorr-similar-weights_01	3009.553	8.87	0.0	0.0	6.78	0.0	2.30e-6	0.0	2.30e-6
eil51_n12_m11_multiple-strongly-corr_01	648.546	4.58	0.0	0.0	6.08	4.6	2.20e-6	4.6	2.20e-6
eil51_n12_m11_uncorr_01	1717.699	3.94	0.0	0.0	7.21	0.0	1.20e-6	0.0	1.20e-6
eil51_n12_m11_uncorr-similar-weights_01	774.107	3.36	0.0	0.0	7.03	0.0	2.30e-6	0.0	2.30e-6
eil51_n12_m55_multiple-strongly-corr_01	1251.780	117.99	0.0	0.0	9.19	0.0	0.0	0.0	0.0
eil51_n12_m55_uncorr_01	8838.012	35.79	0.0	0.0	9.76	0.0	0.0	0.0	0.0
eil51_n12_m55_uncorr-similar-weights_01	3734.895	38.36	12.3	0.0	8.34	12.3	0.0	0.2	0.0
eil51_n15_m14_multiple-strongly-corr_01	547.419	39.82	0.0	0.0	7.87	14.1	1.30e-6	13.3	1.30e-6
eil51_n15_m14_uncorr_01	2392.996	89.46	0.0	0.0	7.28	3.8	0.0	3.8	0.0
eil51_n15_m14_uncorr-similar-weights_01	637.419	16.35	0.0	0.0	6.86	0.0	1.60e-6	0.0	1.60e-6
eil51_n15_m70_multiple-strongly-corr_01	920.372	3984.29	2.1	1.1	12.11	0.0	2.70e-6	0.0	2.70e-6
eil51_n15_m70_uncorr_01	9922.137	740.22	0.0	0.0	9.67	7	1.20e-6	1.9	0.0
eil51_n15_m70_uncorr-similar-weights_01	4659.623	867.78	0.0	0.0	7.98	0.0	0.0	0.0	0.0
eil51_n16_m15_multiple-strongly-corr_01	794.745	105.5	0.0	0.0	7.7	18.9	1.6e-6	18.9	1.6e-6
eil51_n16_m15_multiple-strongly-corr_10	4498.848	623.4	0.0	0.0	9.1	12.9	0.0	16.6	1.3e-6
eil51_n16_m15_uncorr_01	2490.889	59.5	1.0	0.7	8.4	1.6	2.3e-6	1.6	2.3e-6
eil51_n16_m15_uncorr_10	3601.077	211.5	0.0	0.0	9.0	7.1	1.6e-6	7.1	1.6e-6
eil51_n16_m15_uncorr-similar-weights_01	540.897	36.4	0.0	0.0	8.5	0.0	3.0e-6	0.0	3.0e-6
eil51_n16_m15_uncorr-similar-weights_10	3948.211	245.4	0.0	0.0	8.7	5.8	1.5e-6	13.6	0.0
eil51_n17_m16_multiple-strongly-corr_01	685.565	248.6	0.0	0.0	8.4	0.2	1.5e-6	0.0	1.5e-6
eil51_n17_m16_multiple-strongly-corr_10	3826.098	2190.4	0.0	0.0	9.8	0.0	1.5e-6	0.0	1.5e-6
eil51_n17_m16_uncorr_01	2342.664	134.9	0.0	0.0	8.3	0.0	0.0	0.0	0.0
eil51_n17_m16_uncorr_10	2275.279	554.5	0.0	0.0	9.6	0.0	0.0	0.0	0.0
eil51_n17_m16_uncorr-similar-weights_01	556.851	70.8	0.0	0.0	8.1	0.0	0.0	0.0	0.0
eil51_n17_m16_uncorr-similar-weights_10	2935.961	787.7	0.0	0.0	9.7	0.0	0.0	0.0	0.0
eil51_n18_m17_multiple-strongly-corr_01	834.031	715.7	7.9	0.8	10.2	9.2	0.0	12.9	1.7e-6
eil51_n18_m17_multiple-strongly-corr_10	5531.373	6252.4	0.0	0.0	10.5	0.4	1.5e-6	0.4	1.5e-6
eil51_n18_m17_uncorr_01	2644.491	366.1	0.0	0.0	9.7	0.2	0.0	1.8	0.0
eil51_n18_m17_uncorr_10	3222.603	1462.7	0.0	0.0	10.3	0.0	1.3e-6	0.2	0.0
eil51_n18_m17_uncorr-similar-weights_01	532.906	148.3	0.0	0.0	8.5	0.0	1.3e-6	0.0	1.3e-6
eil51_n18_m17_uncorr-similar-weights_10	4420.438	1929.3	0.0	0.0	9.9	0.0	2.9e-6	0.3	1.8e-6
eil51_n19_m18_multiple-strongly-corr_01	910.229	1771.6	0.0	0.0	9.3	20.1	1.6e-6	20.1	1.6e-6
eil51_n19_m18_multiple-strongly-corr_10	-	-	-	-	10.4	-	-	-	-
eil51_n19_m18_uncorr_01	2604.844	830.2	0.0	0.0	9.7	0.0	0.0	0.0	0.0
eil51_n19_m18_uncorr_10	4048.408	3884.3	0.0	0.0	10.9	0.0	1.4e-6	0.0	1.4e-6
eil51_n19_m18_uncorr-similar-weights_01	472.186	412.3	0.0	0.0	9.2	0.0	1.5e-6	0.0	1.5e-6
eil51_n19_m18_uncorr-similar-weights_10	5573.695	5878.8	0.0	0.0	10.5	0.0	0.0	0.0	0.0
eil51_n20_m19_multiple-strongly-corr_01	518.189	4533.7	0.6	0.6	11.1	14.1	1.4e-6	12.3	0.0
eil51_n20_m19_multiple-strongly-corr_10	-	-	-	-	12.1	-	-	-	-
eil51_n20_m19_uncorr_01	2092.673	2456.9	0.0	0.0	8.7	0.0	0.0	0.0	0.0
eil51_n20_m19_uncorr_10	3044.391	12776.0	0.0	0.0	9.8	0.0	0.0	0.0	0.0
eil51_n20_m19_uncorr-similar-weights_01	451.052	1007.7	0.0	0.0	7.9	0.0	0.0	0.0	0.0
eil51_n20_m19_uncorr-similar-weights_10	4169.799	15075.7	0.0	0.0	9.4	0.0	0.0	0.0	0.0

TABLE 7.3: Comparison between DP and the approximate approaches running in 10 minutes limits. Each approximate algorithm runs 10 times for each instance and use the average as the objective Obj. Gap is measured by $\frac{OPT-Obj}{OPT}\%$ and runtime (RT) is in second. The results of C5 and DP-S5 are obtained when they reach the time limit of 10 minutes per instance. Highlighted in blue are the best approximate results.

DP runs out of memory for the instances without results.

Chapter 8

Evolutionary Computation plus Dynamic Programming for the Bi-Objective Travelling Thief Problem

In the previous chapter (Chapter 7), we introduced the exact approaches for the travelling thief problem (TTP) based on the dynamic programming approach for the packing while travelling (PWT, Section 2.3.4) problem introduced in Chapter 6. In this chapter, we will further extend our study to evolutionary algorithms (EAs) that take advantage of the exact approaches. As introduced in Section 3.5, many approximate approaches have been introduced for addressing the TTP. However to the best of our knowledge, all of the existing approaches are focusing on utilising the existing heuristic approaches (such as local search, simulated annealing, tabu search, genetic algorithms, memetic algorithm, swarm intelligence, etc.), incorporating either well-studied operators of the TSP and KP or slight variations of such operators. The heuristic approaches or operators that take advantage of the existing exact algorithms of the TTP [129, 176] are still lacking. On the other hand, very few investigations have been undertaken for the approaches of the multi-objective formulations of the TTP, except by Blank et al. [20] and Yafrani et al. [178].

In this chapter, we consider a bi-objective version of the TTP, where the goal is to minimise the weight as well as to maximise the overall benefit. We present a hybrid approach for the bi-objective TTP that uses a dynamic programming approach for the underlying PWT problem as a subroutine. The evolutionary component of our approach constructs a tour π for the TTP. This tour is then fed into the dynamic programming algorithm to compute a trade-off front for the bi-objective problem. Here the tour π is kept fixed and the resulting packing solutions are Pareto optimal due to the capability of the dynamic programming. A key aspect of the algorithm is to take advantage of the different fronts belonging to different tours for the TTP component, as presumably the global Pareto optimum may contain some segments from the different fronts. Meanwhile, when the evolutionary approach evolves the tours and the current general Pareto front consists of different tours (together with the packing plans), the next challenge is to select tours for the mutations and crossovers that lead to promising new tours. Such tours will result in new Pareto optimal solutions for the overall bi-objective TTP problem, when running the dynamic programming. In short, the selection mechanism will encourage the synergy of the two sub-approaches. We introduce a novel indicator-based evolutionary algorithm (IBEA [182]) that contains a series of customised indicators and parent selections to achieve this goal. Our results show that this approach solves the problem well,

and its by-product, which is the total reward of the single objective TTP, beats the state-of-the-art approach in most cases.

This chapter is based on a paper submitted to a conference [174]. The remainder of the chapter first of all records the bi-objective version of the TTP mathematically, in Section 8.1. Then, Section 8.2 covers the prerequisites required for our approach, which is subsequently introduced in Section 8.3. Section 8.4 provides the detailed description of the computational setup and the analysis of computational experiments. Finally, Section 8.5 draws conclusions from the results in the chapter.

8.1 The Bi-objective Travelling Thief Problem

As introduced in Section 2.3.3 of Chapter 2, the sole objective of the standard single-objective TTP is to find the optimal total reward $\max_{\pi,\rho} \mathcal{Z}(\pi, \rho)$, where

$$\mathcal{Z}(\pi,\rho) = \sum_{i=1}^{n}\sum_{k=1}^{m_i} p_{x_ik}y_{x_ik} - R \cdot \left(\frac{d_{x_n x_1}}{v_{max} - \nu W_{x_n}} + \sum_{i=1}^{n-1} \frac{d_{x_i x_{i+1}}}{v_{max} - \nu W_{x_i}} \right). \tag{8.1}$$

(π, ρ) represents a solution of the TTP, consisting of a tour $\pi = (x_1, \ldots, x_n)$, $x_i \in N$ and packing plan $\rho = (y_{21}, \ldots, y_{nm_n})$, $y_{ik} \in \{0, 1\}$.

Here, we extend the standard formulation of the TTP by introduction of an additional objective function. The new version, named as BO-TTP for short, becomes a bi-objective optimisation problem, where the total accumulated weight

$$\mathcal{W}(\rho) = \sum_{i=1}^{n}\sum_{k=1}^{m_i} w_{ik}y_{ik} \tag{8.2}$$

yields the second criterion. Such extension appears natural regarding the TTP as one may either need to maximise the reward for a given weight of collected items, or determine the least weight subject to bounds imposed on the reward. Note that even if π is fixed, (8.1) is a non-monotone sub-modular function [140] that implies possible deterioration of the reward as the number of selected items, and therefore their total weight, increases. We formulate the BO-TTP as follows:

$$(\pi,\rho) = \begin{cases} \arg\max \mathcal{Z}(\pi,\rho) \\ \arg\min \mathcal{W}(\rho) \end{cases} \quad s.t.\ \mathcal{W}(\rho) \leq C$$

As a bi-objective optimisation problem, BO-TTP asks for a set of Pareto-optimal solutions where each feasible solution cannot be improved in a second objective without degrading quality of the first one, and vice versa. In other words, the goal is to find a set of all non-dominated feasible solutions $X \subseteq \Pi \times P$ such that for any solution $(\pi, \rho) \in X$ there is no solution $(\pi', \rho') \in X$ such that either $(\mathcal{Z}(\pi,\rho) < \mathcal{Z}(\pi',\rho')) \wedge (\mathcal{W}(\rho) \geq \mathcal{W}(\rho'))$ or $(\mathcal{Z}(\pi,\rho) \geq \mathcal{Z}(\pi',\rho')) \wedge (\mathcal{W}(\rho) < \mathcal{W}(\rho'))$ holds, where Π is a set of feasible tours and P is a set of feasible packing plans.

8.2 Prerequisites

As introduced in Section 2.3.4 of Chapter 2, the packing while travelling problem (PWT) is a special case of the TTP, which maximises the total reward for a specific tour π. Thus, an optimal solution of the PWT defines a subset of items producing the maximal gain. This yields a non-linear knapsack problem, which can be efficiently solved via the dynamic programming (DP) approach proposed in Chapter 5. Most importantly, we find that the DP yields not just a single optimal packing plan, but a set of plans $\overline{P}_\pi \subseteq P$, where (π, ρ) and (π, ρ') do not dominate each other for any $\rho, \rho' \in \overline{P}_\pi$. We name the corresponding objective vectors of $\overline{P}_\pi$ as a *DP front*. In Section 8.3, we design our hybrid algorithm that takes advantage of the features of a DP front.

8.2.1 DP Front for a Given Tour

The DP for the PWT introduced in Chapter 5 processes items in the lexicographic order as they appear along a given tour π; that is, item $l \in \pi_i$ strictly precedes item $k \in \pi_j$, to be written as $l \preceq k$, if either $\pi_i < \pi_j$ or $(\pi_i = \pi_j) \wedge (l \leq k)$ holds. Its table B is an $m \times C$ matrix, where entry β_{kw} represents the maximal reward that can be achieved by examining all combinations of items l with $l \preceq k$ leading to the weight equal to w. The base case of the DP with respect to the first item k, according to the precedence order, positioned in node π_i is as follows:

$$\beta_{kw} = \begin{cases} -\frac{R}{v_{max}} \left(\sum_{j=1}^{n-1} d_{\pi_j \pi_{j+1}} + d_{\pi_n \pi_1} \right), & \text{if } w = 0 \\ p_{\pi_i k} - R \left(\sum_{j=1}^{n-1} \frac{d_{\pi_j \pi_{j+1}}}{v_{\pi_j}} + \frac{d_{\pi_n \pi_1}}{v_{\pi_n}} \right), & \text{if } w = w_{\pi_i k} \\ -\infty, & \text{if } w \notin \{0, w_{\pi_i k}\} \end{cases}$$

Here, the first case relates to the empty packing when the thief collects no items at all while travelling along π, and the second computes the reward when only item k is chosen. Where a combination yielding w doesn't exist, $\beta_{kw} = -\infty$. For the general case, let item l be the predecessor of item k with regard to the precedence order. And let $\beta(k\cdot)$ denote the column containing all the entries β_{kw} for $w \in [0, C]$. Then based on $\beta(l\cdot)$ one can obtain $\beta(k\cdot)$ computing each entry β_{kw}, assuming that item k is in node π_i, as

$$max \begin{cases} \beta_{lw} \\ \beta_{lw - w_{\pi_i k}} + p_{\pi_i k} - R \sum_{j=i}^{n-1} \left(\frac{d_{\pi_j \pi_{j+1}}}{v_{max} - \nu w} - \frac{d_{\pi_j \pi_{j+1}}}{v_{max} - \nu (w - w_{\pi_i k})} \right) \\ \quad - R \left(\frac{d_{\pi_n \pi_1}}{v_{max} - \nu w} - \frac{d_{\pi_n \pi_1}}{v_{max} - \nu (w - w_{\pi_i k})} \right) \end{cases}$$

In order to reduce the search space, in each column the cells dominated by other cells are to be eliminated, i.e. if $\beta_{kw_1} > \beta_{kw_2}$ and $w_1 \leq w_2$, then $\beta_{kw_2} = -\infty$. An optimal solution derived by the DP corresponds to the maximal reward stored in

the last column of B. That is, $\max_w \{\beta(s, w)\}$ is the value of an optimal solution, where s is the last item according to the precedence order.

The last column of B can be considered as a complete set of non-dominated packing plans $\overline{P}_\pi \subseteq P_\pi \subseteq P$, where P_π is the set of all feasible packing plans for a given tour π. The packing plans in $\overline{P}_\pi$ are non-dominated exclusion of any dominated solutions during the solution construction process.

Definition 2. *Letting τ and T_π be the corresponding objective vectors sets of $\overline{P}_\pi$ and P_π respectively, τ is the Pareto front of T_π. We therefore name τ as a* DP front *for the given tour π.*

A DP front τ for a tour π is a complete non-dominated set, as it contains all non-dominated objective vectors in T_π. We take advantage of this completeness to generate the spread of solutions in our bi-objective approach in Section 8.3.

8.2.2 Generation of Multiple DP Fronts

As a single DP front τ is produced for a single given tour π, i.e. $\pi \mapsto \tau$, we could generate multiple TSP tours to get a set of DP fronts. In practice, various algorithms are capable of producing superior tours for the TSP, and therefore many approaches to the TTP use this capability to succeed. High-performing TTP algorithms are commonly two-stage heuristic approaches, like those proposed by Polyakovskiy et al. [139], Faulkner et al. [56], and El Yafrani et al. [50]. Specifically, their first step generates a near-optimal TSP tour and the second step completes solution by selection of a subset of items. Most of the approaches utilise the Chained Lin-Kernighan heuristic [5], because it is able to provide very tight upper bounds for TSP instances in short time. The knapsack component then is often handled via constructive heuristics or evolutionary approaches. However, the TTP is essentially structured in such way that the importance of its both components is almost equal within the problem. Although near-optimal TSP solutions can give good solutions to the TTP, most of them are far away from being optimal [176]. This is the reason for our first experimental study here, where we investigate the impact of several TSP algorithms on TTP solutions. Note that owing to the DP we are able to solve the knapsack part to optimality, which contributes to the validity of our findings.

We analysed five algorithms for the TSP: the Inver-over heuristic (INV) [164], the exact solver Concorde (CON) [4], the ant colony-based approach (ACO) [45], the Chained Lin-Kernighan heuristic (LKH) [5] and its latest implementation (LKH2) [73]. We ran each algorithm $10,000$ times on every instance of the *eil76* series of the TTP benchmark suite [139]. We computed 100 (capped due to practical reasons) distinct tours by INV, 25 by CON, 24 by the both ACO and LKH, and 12 by LKH2. The lengths of the tours generated by INV are narrowly distributed around the average of 588.64 with the standard deviation being 2.55. By contrast, every other algorithm generates tours having the identical tour length of 585, which beats INV.

We then applied the DP to every tour produced by each of the algorithms. Figure 8.1 depicts the resulted rewards on some sample TTP instances, where each box with whiskers reports the distribution of the rewards for a certain instance and the corresponding algorithm. The central mark of each box indicates the median of

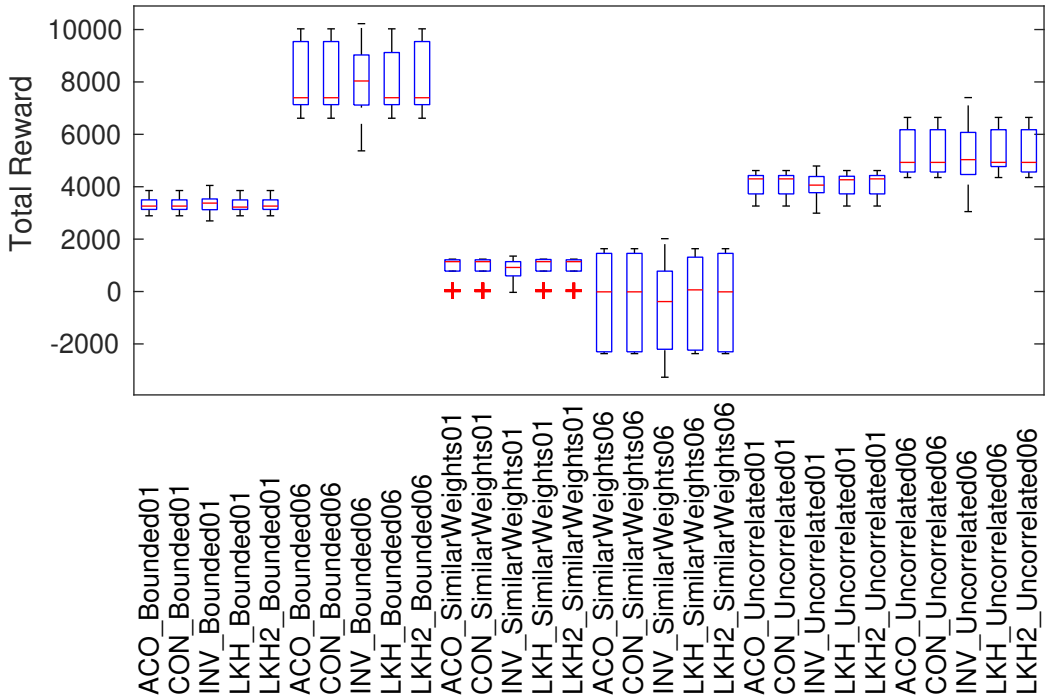


FIGURE 8.1: Exploring diversity of TSP tours on the *eil76_n75* series of the TTP instances.

rewards, and the bottom and top edges of the box indicate the 25th and 75th percentiles, respectively. The whiskers extend to the most extreme rewards without considering outliers, and outliers are plotted individually as plus signs. From the plot, we may observe that the tours generated by the CON, ACO, LKH and LKH2 have similar distributions of rewards. By contrast, the boxes of INV seem to be more extreme on the both sides. This means that the distribution of rewards via INV is more diverse and the best of the rewards outperform the others. In other words, though the Inver-over heuristic may lose against modern TSP approaches, it performs better in the role of generator of varied tours for the TTP. It may act as a seeding algorithm for a population in evolutionary algorithms.

In Figure 8.2, we visualise the collection of the DP fronts produced by the DP on the TTP instance *eil76_n75_uncorr_01* [139]. The corresponding tours are the 100 tours generated by the Inver-over heuristic. Actually, the plot depicts 200 fronts since the DP was applied to a tour and its reversed order.

Definition 3. *Given n DP fronts $\tau_1, \ldots, \tau_n$, let Φ denote a union of the fronts as $\Phi = \cup_{i=1}^{n} \tau_i$. Then a subset $\omega \subseteq \Phi$ is the Pareto front of Φ called as the* surface *of Φ.*

The surface ω is formed by the union of all superior points resulted from different DP fronts in Φ. It is further used to guide evolution process in our approach.

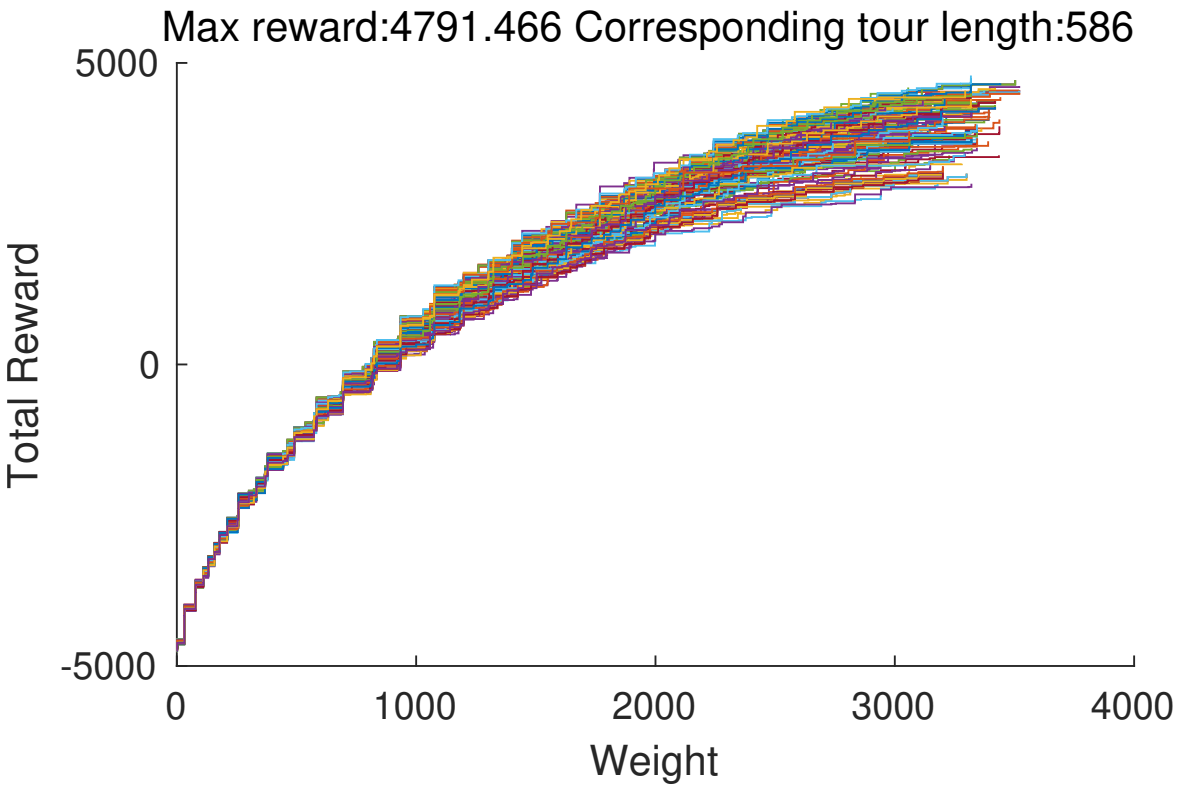


FIGURE 8.2: The visualisation of 200 DP fronts, generated according to 100 TSP tours produced by Inver-over for the TTP instance *eil76_n75_uncorr_01*.

8.3 A hybrid evolutionary approach

Multi-objective optimisation algorithms guided by evolutionary mechanisms explore the decision space iteratively in order to determine a set of Pareto-optimal solutions. Indeed, many of them may act myopically as they sample the space searching for individual solutions without clear vision of the whole picture in terms of other solutions and their number. Therefore, achieving strong diversity in exploring the space plays an important role in evolutionary algorithm design. Herein we discuss one way to overcome potential issues related to diversity and propose a hybrid approach where evolutionary techniques and dynamic programming find synergy in their combination.

One of the challenges of multi-objective optimisation is to keep the wide spread of solutions, which has to be guaranteed by strong diversity. Modern approaches normally incorporate additional processes to tackle this, such as the density estimation and/or crowdedness-comparison operator in SPEA2 [183] and NSGA-II [42]. In our approach, the DP is incorporated as a subroutine capable of producing at once a series of possible decisions with regard to a given tour. Thus, when a tour is specified, the DP guarantees that a corresponding front will be built without missing any of its points due to the completeness of the DP front, which thus also guarantees a good spread of solutions.

On the other hand, due to the typically observed non-dominance of single DP

fronts, the global Pareto optimality of the BO-TTP may be formed either by a single DP front or by the combination of segments from different top DP fronts. In Figure 8.2, we may observe that the DP fronts are all intertwined together, including the ones at the surface of the fronts collection. This seems to indicate that the Pareto-optimal set of solutions is more likely to be the result of multiple TSP tours and their DP fronts. We would like our evolutionary mechanism to take advantage of this and to keep the top DP fronts so as to improve the population further. In order to achieve this as well as to overcome the drawback of existing multi-objective evolutionary optimisation algorithms that focus on individual solutions, we design our hybrid IBEA with particular indicators and selection mechanisms in orchestrating improvement of Pareto front guided by the information of the DP fronts for most promising TSP tours.

Our hybrid approach reduces the search space to some extent by decomposing the problem and thus transforming it. Evolutionary optimisation approaches traditionally depend on the choice of solution encoding (i.e. chromosome). Our approach treats a single TSP tour as an individual. Thus, a set of tours yields a population. Indeed, it operates on a reduced set of variables (implying shorter chromosomes), thus decreasing memory consumption and the number of internally needed sorting operations, comparisons and search operations, as it does not explicitly include binary variables required for the packing part. This approach is in fact a type of two-level encoding [177], by which the individuals are encoded in the upper-level (tours), and evaluated by solving the lower-level (packing plan).

Algorithm 17 Hybrid IBEA Approach

Input: population size μ; limit on the number of generations α;
Initialisation:
set the iteration counter $c = 0$;
populate $\overline{\Pi}$ with μ new tours produced by the TSP solver;
while $(c \leq \alpha)$ **do**
  set $c = c + 1$;
  Indicator:
  run the DP for every tour $\pi \in \overline{\Pi}$ to compute its DP front τ;
  apply indicator function $\mathcal{I}(\tau)$ to calculate the indicator value for every individual tour $\pi \in \overline{\Pi}$;
  Survivor Selection:
  repeatedly remove the individual with the smallest indicator value from the population $\overline{\Pi}$ until the population size is μ (ties are broken randomly);
  Parent Selection:
  apply parent selection procedure to $\overline{\Pi}$ according to the indicator values to choose a set Λ of λ parent individuals;
  Mating:
  apply crossover and mutation operators to the parents of Λ to obtain a child population Λ';
  set the new population as $\overline{\Pi} = \overline{\Pi} \cup \Lambda'$;
end while

Algorithm 17 sketches the whole approach, which we adopted from the original

IBEA introduced by Zitzler et al. [182]. It accepts μ as a control parameter for the size of the population $\overline{\Pi} \subseteq \Pi$ and α as a limit on the number of iterations, which defines its termination criterion. In order to utilise the information within the DP fronts to guide the evolution of individual tours, we design new indicators to be computed based on the DP fronts instead of directly on the individuals. Our specific selection mechanisms then filter the individuals according to the indicator values in order to find the tours with better DP fronts.

The rest of this section first introduces the indicator functions we apply to TSP solutions. Next, it details a parent selection mechanism to mate existing individuals from the population. It ends with a discussion of mutation and crossover operators guiding the search.

8.3.1 Design of Indicators

The designs of our indicators are based on the idea of measuring how each DP front contributes to the surface ω of the fronts' union Φ corresponding to the population $\overline{\Pi}$. The surface ω introduced in Definition 8.2.2 is the union of all best segments from different DP fronts in Φ. Given a DP front τ for a tour $\pi \in \overline{\Pi}$ and a measurement function $\mathcal{M}$ of a front, we use the followed formula to calculate the indicator $\mathcal{I}$:

$$\mathcal{I}(\tau) = 1 - \frac{\mathcal{M}(\omega \setminus \tau)}{\mathcal{M}(\omega)}. \tag{8.3}$$

This formula measures how much we could lose (expressed as a value from 0 to 1) if we did not include the segments of the front τ to the surface ω, i.e. $\omega \setminus \tau$. In the following, we study two types of the measurement functions: Surface Contribution (SC) and Hypervolume (HV), hence two corresponding indicators: the Loss of Surface Contribution (LSC) and the Loss of Hypervolume (LHV).

Loss of Surface Contribution. Our first indicator is Surface Contribution (SC), which is a novel and direct measure. Given the union of a set of fronts Φ, a front $\tau \subseteq \Phi$ and the surface $\omega \subseteq \Phi$, $SC(\tau)$ counts the number of objective vectors that τ contributes to ω, as defined by:

$$SC(\tau) = \frac{|\omega \cap \tau|}{|\omega|}. \tag{8.4}$$

Using SC (8.4) to replace the $\mathcal{M}$ function in (8.3), we have the formula of LSC as follows:

$$LSC(\tau) = 1 - SC(\omega \setminus \tau).$$

Loss of Hypervolume. In multi-objective optimisation, the hypervolume indicator is a traditional indicator used to indicate the quality of a set of objective vectors [184]. In the bi-criteria case, when a front is given as a set of points in two-dimensional space, its value is computed as a sum of areas of rectangular regions.

Let $(0, C)$ be the reference point for our problem, which implies that only the range of non-negative objective values is taken into account. In addition, let $p = (u, v) \in \tau$ be a bi-dimensional objective vector in a DP front τ while $u > 0$ and

$v < C$, $HV(\tau)$ calculates the hypervolume for τ as:

$$HV(\tau) = \sum_{p \in \tau} u_p \left(v_p - v_{p-1}\right)$$

Putting $HV(\tau)$ back to (8.3), we have the loss of hypervolume $LHV\left(\tau\right)$ computed as

$$LHV\left(\tau\right) = 1 - \frac{HV(\omega \setminus \tau)}{HV(\omega)}.$$

8.3.2 Parent Selection Mechanisms

With the individuals in the population $\overline{\Pi}$ being measured by the defined indicators, we can study strategies that shall efficiently select good individuals. There are five parent selection schemes that we take into consideration due to their popularity or previous theoretical findings. In comparison, we introduce two simple and arbitrary selections as well as a traditional policy to be a baseline. In this study, we expect to find a well-performing combination of indicator and selection to encourage the synergy of the DP and evolutionary approach.

Rank-based Selection (RBS). In the rank-based selection policy, individuals are first ranked with respect to the value of an indicator. The selection policy is based then on a specific distribution law affecting the choice of a parent. Here, we study three schemes introduced by Osuna et al. [135], namely exponential (EXP), inverse quadratic (IQ) and Harmonic (HAR), and make them a part of our hybrid approach. Given a population of size μ, the probability of selecting the ith ranked individual according to EXP, IQ and HAR is, respectively,

$$\frac{2^{-i}}{\sum_{j=1}^{\mu} 2^{-j}}, \ \frac{i^{-2}}{\sum_{j=1}^{\mu} j^{-2}}, \ \frac{i^{-1}}{\sum_{j=1}^{\mu} j^{-1}}. \tag{8.5}$$

Fitness-Proportionate Selection (FPS). This rule estimates an individual $\pi \in \overline{\Pi}$ according to the indicator $\mathcal{I}(\tau)$ of its DP front τ. It has the following form:

$$FPS\left(\pi_i\right) = \frac{\mathcal{I}(\tau_i)}{\sum_{j=1}^{\mu} \mathcal{I}(\tau_j)}. \tag{8.6}$$

Tournament Selection (TS). This policy applies the tournament selection [121], but employs indicators discussed in Section 8.3.1 to rank individuals.

Arbitrary Selection (AS). Here, we consider two different rules: the best arbitrary selection (BST) and another one, which we call extreme (EXT). The former ranks individuals of a population with accordance to the value of an indicator and selects the best half of the population. The latter proceeds similarly selecting 25% of the best and 25% of the worst individuals.

Uniformly-at-random Selection (UAR). This traditional policy selects a parent from a population with probability $\frac{1}{\mu}$ uniformly at random.

8.3.3 Mutation and Crossover Operators

In our approach, we adopt a multi-point crossover operator that has already proved its efficiency for the TTP in [50]. As an (un-optimised) rule, we perform the crossover operation on a tour with 80% probability. It is always followed by the mutation procedure, which either applies the classical 2OPT mutation [38] or re-inserts a node to another location. Both the node and the location are selected uniformly at random. We name these two operators 2OPT and JUMP, respectively.

8.4 Computational Experiments

8.4.1 Computational Set Up

We examine the IBEA presented in Algorithm 17 by going through each of the two indicators and the eight parent selections, resulting in a total of 16 settings. Each setting represents a combination of one of the 2 indicators and one of the 8 parent selections respectively. For example, FPS on LHV means the combination of the FPS selection and the LHV indicator.

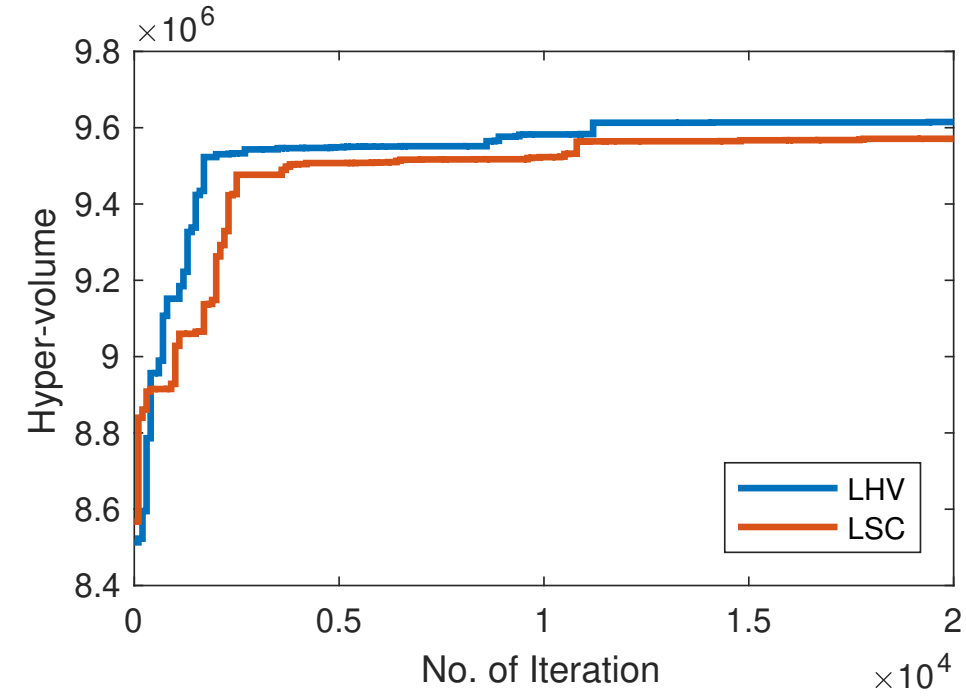
From the original set of TTP instances, we use three different types, namely bounded-strongly-correlated (*Bounded*), uncorrelated (*Uncorrelated*) and uncorrelated-with-similar-weights (*SimilarWeights*), selected from three instance series: *eil51*, *eil76*, *eil101* in the TTP benchmark [139]. We run our approach 30 times repetitively on each selected instance. Each time, the algorithm runs 20,000 generations on a population $\overline{\Pi}$ in size of 50. Figure 8.3 demostrates the convergence curve on the eil76_n75_uncorr_01 instance.

Due to the significant computing cost, our experiments run on the supercomputer in our university, which consists of 5568 Intel(R) Xeon(R) 2.30GHz CPU cores and 12TB of memory. We allocate 16 CPU cores and 16GB of memory to each individual experiment. In total, 69,120 CPU cores and the same number of GB memory have been allocated for the experiments. Overall the experiments consumed around 170,000 CPU-hours.

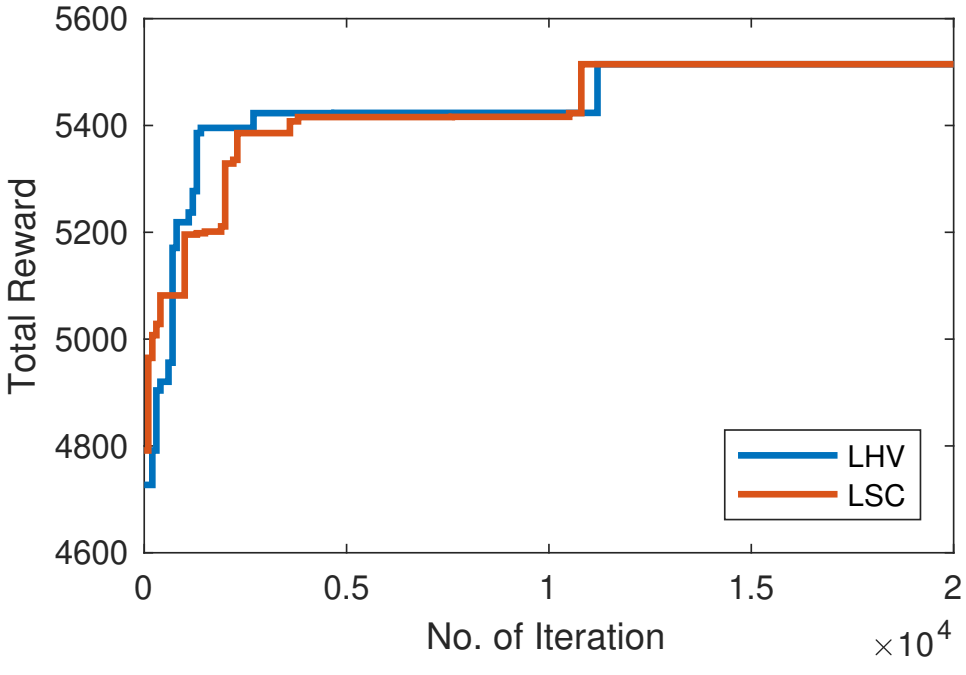
8.4.2 Results and Analysis

To compare the outcomes of the different approaches based on the final populations $\overline{\Pi}$ of tours, we calculate the hypervolumes for the surface of resulting non-dominated solutions. We also store the corresponding total reward in order to compare with the results from the state-of-the-art single-objective approach: MA2B [50] (see comparison in [169]).

However, due to the varied mean values and unknown global optima of different TTP instances, it is hard to analyse and compare across instances. Nevertheless, such a comparison is desired because such analysis or comparison may provide a more general view for our algorithm. We design a statistical comparison to overcome this as follows. Firstly, we choose the uniformly-at-random (UAR) selection as the baseline, which creates two baseline settings, namely UAR on LHV and UAR on LSC. We secondly conduct *Welch's t-test* [170] between the results of the others and the baselines for two indicators respectively.



(a) Hypervolume



(b) Total Reward

FIGURE 8.3: Convergence curve on the eil76_n75_uncorr_01 instance.

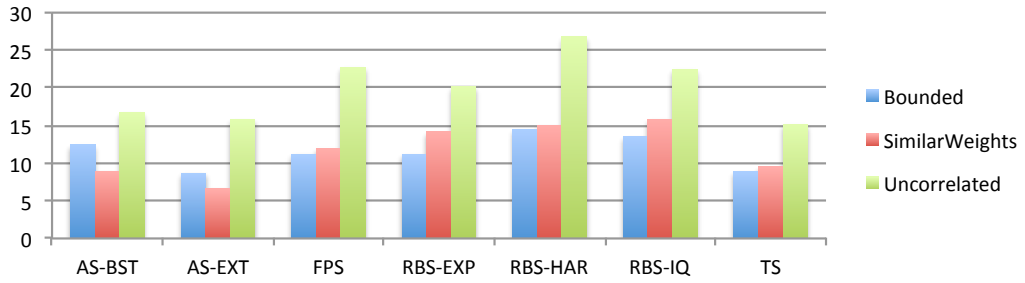
The results of the t-test are probability values (p-values), each of which measures the likelihood of one selection to the corresponding baseline with respect to their performance. For example, we have the p-value being 4.75×10^{-7} in the case of comparing the hypervolume of the FPS and the UAR on LHV. This means that the probability of the FPS performing identical to the UAR on LHV (as expressed by having the same means) is less than 0.0000475%. In fact, the former performs much better than the latter on average. In order to improve the readability, we use the logarithm of the p-value in our plots. Thus, the measure of the FPS on LHV in our little example is 6.32 (i.e. $\log_{10}(4.75 \times 10^{-7})$). In short, the larger the logarithmic p-value is, the better the selection is against the UAR.

Figure 8.4 depicts the overall results of the Welch's t-test, in which we categorise our results into three types of bars according to three types of TTP instances: Bounded, Uncorrelated and SimilarWeights. Each bar in the plots represents the mean of the logarithmic p-values of several instances in this category, for example *eil51_n50_bounded-strongly-corr_01.ttp*, *eil76_n75_bounded-strongly-corr_01.ttp* and the *eil101_n100_bounded-strongly-corr_01.ttp*. From it we may observe distinguishable patterns between the selections running on the LHV and the LSC respectively. For example, the three rank-based selection (RBS) schemes generally perform better on LHV than on LSC, among which the HAR is the best. According to the definitions, the HAR is the least aggressive scheme among the three, with a fat tail and relatively small probability for selecting the best few individuals [135]. It seems to imply that the LHV benefits more from the diversity of candidates. By contrast, the AS-BST performs best on LSC, which might imply that the LSC relies more on a few outstanding individuals for approximating, as the AS-BST only focuses on the best ones.

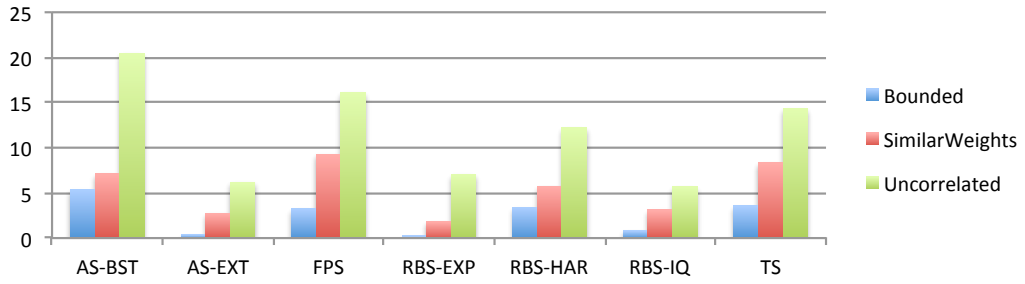
In terms of different types of TTP instances, we may observe that the IBEA performs best on the uncorrelated instances in all of the settings, while being worst on the strongly bounded ones in most of the settings. This to some extent supports the conjecture that strongly bounded TTP instances are the (relatively) hard ones and uncorrelated instances are the easy ones [139].

With regard to the choice of the parent selections, besides the RBS-HAR and the AS-BST which perform best on LHV and LSC respectively, we would like to recommend the FPS as well. This selection seems to be the safest choice, as it performs consistently well on different settings.

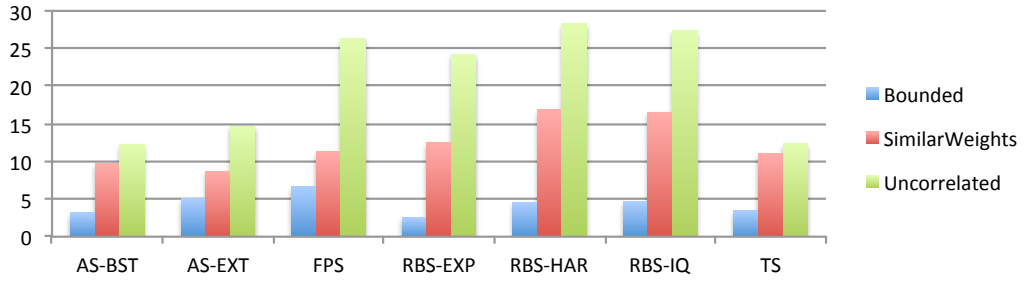
Overall, we may observe from Figure 8.4 that the figures of the hypervolume generally agree with those of the total reward. This somewhat suggests that optimising the bi-objective TTP brings good results for the single objective TTP as well. We do further comparison between our approach and the state-of-art algorithm of the single objective TTP, namely MA2B [50], to verify this conjunction. In order to be more compellable, we choose safest parent selection, i.e. the FPS, instead of the aggressive ones. For the MA2B, we run it 30 times repetitively for each case with the identical time limits to our approach. The results in Table 8.1 show that in the majority of the test cases, our approach yields better rewards comparing to the MA2B for the corresponding single objective TTP, while we are optimising the bi-objective formula.



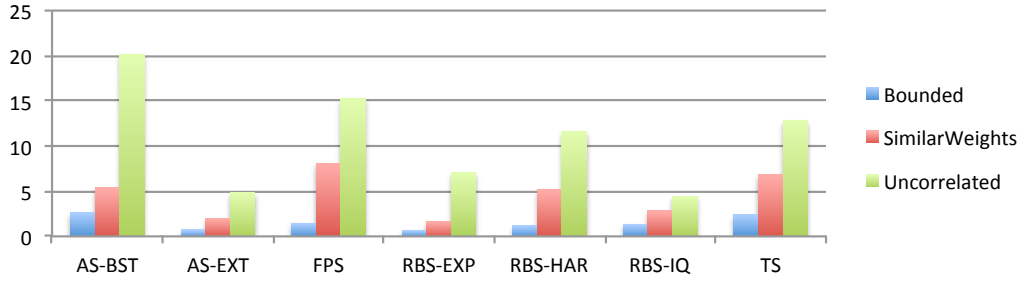
(a) Hypervolume of each selection on LHV



(b) Hypervolume of each selection on LSC



(c) Total Reward of each selection on LHV



(d) Total Reward of each selection on LSC

FIGURE 8.4: The sum of the logarithm of the p-values of performing Welch's t-test for the selections each respectively against the UAR selection.

MA2B				
		Mean	Max	SD
eil51_n50	Uncorrelated	2805.000	2855	27.814
	SimilarWeights	1416.348	1460	47.906
	Bounded	4057.652	4105	25.841
eil76_n75	Uncorrelated	5275.067	5423	78.138
	SimilarWeights	1398.867	1502	55.448
	Bounded	3849.067	4109	139.742
eil101_n100	Uncorrelated	3339.600	3789	388.360
	SimilarWeights	2215.500	2483	235.905
	Bounded	4949.000	5137	139.285
FPS LHV				
		Mean	Max	SD
eil51_n50	Uncorrelated	2828.728	2854.543	15.357
	SimilarWeights	1413.044	1459.953	17.780
	Bounded	4229.149	4230.997	10.118
eil76_n75	Uncorrelated	5445.624	5514.666	58.992
	SimilarWeights	1477.680	1513.404	24.494
	Bounded	4042.449	4108.760	38.805
eil101_n100	Uncorrelated	3620.844	3943.425	222.815
	SimilarWeights	2431.907	2482.462	52.265
	Bounded	5094.246	5233.513	65.267
FPS LSC				
		Mean	Max	SD
eil51_n50	Uncorrelated	2810.509	2832.496	18.076
	SimilarWeights	1426.135	1459.953	21.990
	Bounded	4231.299	4241.199	1.881
eil76_n75	Uncorrelated	5392.575	5514.666	73.029
	SimilarWeights	1474.803	1513.404	21.346
	Bounded	4054.815	4102.167	21.440
eil101_n100	Uncorrelated	3664.369	3846.172	124.994
	SimilarWeights	2436.374	2482.462	49.731
	Bounded	5067.070	5233.513	55.587

TABLE 8.1: Comparison of the total reward between running the state-of-art approach MA2B and the IBEA with the selection being FPS and the indicator being LHV and LSC respectively. Each approach runs 30 times on the TTP instances. Highlighted are the results that are better than MA2B.

8.5 Conclusion

In this chapter, we have investigated a new bi-objective travelling thief problem which optimises both the total reward and the total weight. We have proposed a hybrid indicator-based evolutionary algorithm (IBEA) that utilises the exact dynamic programming algorithm for the underlying PWT problem as a subroutine to evolve the individuals. This approach guarantees the spread of solutions, without introducing additional spread mechanisms. Furthermore, we have designed and studied novel indicators and selection schemes that take advantage of the information in the Pareto fronts generated by the exact approach for evolving solutions towards the global Pareto optimality. Our results show that this approach solves the problem well, because its by-products, which are the results for the single-objective travelling thief problem, beat the state-of-the-art approach.

Chapter 9

Conclusions

In this thesis, we presented our investigations for complex and multi-component optimisation problems, especially the travelling thief problem (TTP). The TTP is a benchmark problem inspired by the multi-component optimisation in logistics and supply chain management. Our focus of attention was on the motivations, insights and approaches.

The context of our investigations is described in Chapters 2 and 3. We introduced the general optimisation problems and motivation to solve the multi-component optimisation problems by using two real-world cases in Chapter 2. One of them is our case-study for the renewable energy industry, and the other is one of the original inspirations of the TTP. Furthermore, the mathematical definition of the TTP was introduced, as well as state-of-the-art development of the TTP, including existing approximate approaches for the TTP and a relevant problem: the packing while travelling (PWT) problem. In Chapter 3, diverse approaches for optimisation problems were discussed and some examples were given in terms of exact and heuristic approaches, as well as multi-objective optimisation.

In Chapter 4, we elaborated the case-study in the renewable energy industry, which provides insights into the arrays of submerged wave energy converters (WECs). WECs capture energy from the movement of waves in order to generate electricity or to pump water onshore to create potable water by reverse osmosis. The interactions of each individual WEC in an array have a significant impact on the overall power absorption of the farm, which demonstrates how complex or multi-component systems operate in the real-world. Hence optimising each WEC individually cannot yield overall optimality. A second major obstruction to optimisation is that the simulations of the interactions are computationally prohibitively expensive, taking hours or even days to complete. Through model approximations and caching, we achieved up to 350-fold speed-ups in the simulation times needed, which in turn allowed us to optimise the interactions in WEC arrays iteratively. Our approach for tackling this problem is particularly problem-specific, which, to some extent, motivates a more general way of investigating the multi-component optimisation problems.

Within Chapter 5, we have paid special attention to the PWT problem, which is a simplified version of the TTP, with the tour predefined. The PWT problem is similar to the classical 0-1 knapsack problem but is more complex due to the nonlinearity introduced by the TTP. Hence the investigations into it also provide better insights into the TTP. With regard to the study of the renting rate that connects the profit and the cost part of the problem, we have shown through theoretical and experimental investigations how the renting rate affects the number of items that can be

eliminated by the simple pre-processing scheme. Furthermore, we have constructed instances in a theoretical and experimental way where a simple baseline $(1 + 1)$ EA fails to obtain an optimal solution.

Our study and examination of the non-linear PWT problem continued in Chapter 6. With a better understanding of the nonlinearity of the PWT problem, a dynamic programming algorithm has been designed to solve the problem in pseudo-polynomial time. The algorithm later became the key to better understanding the TTP and to form hybrid algorithms, combining with meta-heuristics, to solve the TTP. We have shown that the original objective of the problem is hard to approximate and have given a fully polynomial approximation scheme (FPTAS) for optimising the amount that can be gained for the smallest possible travel cost. Our experimental results on different types of instances show the advantage of the dynamic programming over the previous approach, based on mixed integer programming and branch-infer-and-bound concepts. On the other hand, the FPTAS demonstrated its effectiveness for instances with a large weight and profit range in our specially designed experiments.

Despite the fact that the TTP has attracted significant attention in recent years within the Evolutionary Computation community and many approximate approaches have been proposed for it, exact approaches are still lacking. In Chapter 7, we have presented and evaluated our exact approaches for the TTP based on dynamic programming, branch and bound method, and constraint programming. In addition, the exact solutions provided by our DP approach have been used to evaluate the performance of current state-of-the-art TTP approximate approaches. Our investigations show that, in most cases, they are obtaining close to optimality, while for a small fraction of tested instances, a gap to the optimal solutions of more than 10% is observed.

In Chapter 8, we proposed a novel indicator-based hybrid evolutionary approach that combines approximate and exact algorithms. We applied it to a new bi-criteria formulation of the TTP. Our approach employs the exact dynamic programming algorithm for the underlying PWT problem as a subroutine within a bi-objective evolutionary algorithm. This design takes advantage of the data extracted from the Pareto fronts generated by the dynamic program to achieve better solutions. Furthermore, we developed a number of novel indicators and selection mechanisms to strengthen the synergy between the two algorithmic components of our approach. The results of the computational experiments show that the approach is capable of outperforming the state-of-the-art results from the single-objective counterpart.

In general, our studies contribute theoretical insights of the TTP, which help to form the exact and heuristic hybrid approaches. However, compared with the TSP, our best exact approach can only solve very small instances, and more efficient heuristic operator of TTP is yet to be found. The future work certainly lies on the direction on finding better exact and approximate approaches or operators. Meanwhile, our study on dynamic programming and FPTAS demonstrate their efficiency on the underlying knapsack problem of the TTP, which might be further investigated in terms of the hybrid approaches with heuristics similar to our proposed one.

Bibliography

[1] E. P. Adorio and U. Diliman. "Mvf-multivariate test functions library in c for unconstrained global optimization". In: *Quezon City, Metro Manila, Philippines* (2005), p. 44.

[2] A. de Andrés, R. Guanche, L. Meneses, C. Vidal, and I. Losada. "Factors that influence array layout on wave energy farms". In: *Ocean Engineering* 82 (2014), pp. 32–41.

[3] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton, NJ, USA: Princeton University Press, 2007.

[4] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. "Concorde TSP solver". In: (2006). URL: http://www.math.uwaterloo.ca/tsp/concorde.html.

[5] D. Applegate, W. J. Cook, and A. Rohe. "Chained Lin-Kernighan for Large Traveling Salesman Problems". In: *INFORMS Journal on Computing* 15.1 (2003), pp. 82–92. DOI: 10.1287/ijoc.15.1.82.15157. URL: https://doi.org/10.1287/ijoc.15.1.82.15157.

[6] A. Auger and B. Doerr. *Theory of Randomized Search Heuristics: Foundations and Recent Developments*. World Scientific Publishing Co., Inc., 2011.

[7] A. Babarit. "On the park effect in arrays of oscillating wave energy converters". In: *Renewable Energy* 58 (2013), pp. 68–78.

[8] T. Bäck, D. Fogel, and Z. Michalewicz, eds. *Handbook of Evolutionary Computation*. New York, NY: Institute of Physics Publishing Ltd, Bristol and Oxford University Press, 1997.

[9] Y. Bar-Yam. "Engineering Self-Organising Systems". In: ed. by S. A. Brueckner, G. Marzo Serugendo, A. Karageorgos, and R. Nagpal. Berlin, Heidelberg: Springer-Verlag, 2005. Chap. About Engineering Complex Systems: Multiscale Analysis and Evolutionary Engineering, pp. 16–31. ISBN: 978-3-540-26180-3. URL: http://dl.acm.org/citation.cfm?id=2167657.2167660.

[10] D. M. Bates and D. G. Watts. *Nonlinear regression analysis and its applications*. English. Originally published in case binding: c1988. New York ; Chichester : Wiley, 2007. ISBN: 9780470139004 (pbk.)

[11] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear programming - theory and algorithms (2. ed.)* Wiley, 1993. ISBN: 978-0-471-55793-7.

[12] J. E. Beasley. "OR-Library: Distributing Test Problems by Electronic Mail". In: *The Journal of the Operational Research Society* 41.11 (1990), pp. 1069–1072. ISSN: 01605682, 14769360. URL: http://www.jstor.org/stable/2582903.

[13] S. Bellew, T. Stallard, and P. Stansby. "Optimisation of a heterogeneous array of heaving bodies". In: *Proceedings of the 8th European Wave and Tidal Energy Conference*. 2009, pp. 519–527.

[14] R. Bellman. *Dynamic Programming*. Princeton, NJ, USA: Princeton University Press, 2010.

[15] P. Benchimol, W.-J. v. Hoeve, J.-C. Régin, L.-M. Rousseau, and M. Rueher. "Improved filtering for weighted circuit constraints". In: *Constraints* 17.3 (July 2012), pp. 205–233. ISSN: 1572-9354. DOI: 10.1007/s10601-012-9119-x.

[16] D. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1999.

[17] D. P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods (Optimization and Neural Computation Series)*. 1st ed. Athena Scientific, 1996. ISBN: 1886529043.

[18] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. 1997.

[19] J. M. Bishop. "Stochastic searching networks". In: *1989 First IEE International Conference on Artificial Neural Networks, (Conf. Publ. No. 313)*. Oct. 1989, pp. 329–331.

[20] J. Blank, K. Deb, and S. Mostaghim. "Solving the Bi-objective Traveling Thief Problem with Multi-objective Evolutionary Algorithms". In: *Evolutionary Multi-Criterion Optimization: 9th International Conference, EMO 2017, Münster, Germany, March 19-22, 2017, Proceedings*. Ed. by H. Trautmann, G. Rudolph, K. Klamroth, O. Schütze, M. Wiecek, Y. Jin, and C. Grimme. Cham: Springer International Publishing, 2017, pp. 46–60. ISBN: 978-3-319-54157-0. DOI: 10.1007/978-3-319-54157-0_4. URL: https://doi.org/10.1007/978-3-319-54157-0%5C_4.

[21] C. Blum and A. Roli. "Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison". In: *ACM Comput. Surv.* 35.3 (Sept. 2003), pp. 268–308. ISSN: 0360-0300. DOI: 10.1145/937503.937505. URL: http://doi.acm.org/10.1145/937503.937505.

[22] C. Blum and A. Roli. "Metaheuristics in combinatorial optimization: Overview and conceptual comparison". In: *ACM Comput. Surv.* 35.3 (2003), pp. 268–308. DOI: 10.1145/937503.937505. URL: http://doi.acm.org/10.1145/937503.937505.

[23] M. R. Bonyadi and Z. Michalewicz. "Evolutionary Computation for Real-World Problems". In: *Challenges in Computational Statistics and Data Mining*. Ed. by S. Matwin and J. Mielniczuk. Vol. 605. Studies in Computational Intelligence. Springer, 2016, pp. 1–24. ISBN: 978-3-319-18780-8. DOI: 10.1007/978-3-319-18781-5_1. URL: http://dx.doi.org/10.1007/978-3-319-18781-5_1.

[24] M. R. Bonyadi, Z. Michalewicz, and L. Barone. "The travelling thief problem: The first step in the transition from theoretical problems to realistic problems". In: *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2013, Cancun, Mexico, June 20-23, 2013*. IEEE, 2013, pp. 1037–1044. ISBN: 978-1-4799-0452-5. DOI: 10.1109/CEC.2013.6557681. URL: https://doi.org/10.1109/CEC.2013.6557681.

[25] M. R. Bonyadi, Z. Michalewicz, M. R. Przybylek, and A. Wierzbicki. "Socially Inspired Algorithms for the Travelling Thief Problem". In: *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. GECCO '14. Vancouver, BC, Canada: ACM, 2014, pp. 421–428.

[26] L. T. Bui and S. Alam. *Multi-Objective Optimization in Computational Intelligence: Theory and Practice (Premier Reference Source)*. 1st ed. Hershey, PA, USA: IGI Global, 2008. ISBN: 1599044986.

[27] V. Černý. "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm". In: *Journal of Optimization Theory and Applications* 45.1 (Jan. 1985), pp. 41–51. ISSN: 1573-2878. DOI: 10.1007/BF00940812. URL: https://doi.org/10.1007/BF00940812.

[28] A. Chertok. *Wave-actuated power take-off device for electricity generation*. Report. Resolute Marine Energy, 2013.

[29] B. Child and V. Venugopal. "Optimal configurations of wave energy device arrays". In: *Ocean Engineering* 37.16 (2010), pp. 1402–1417.

[30] N. Christofides. *Worst-case analysis of a new heuristic for the travelling salesman problem*. Technical Report 388. Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

[31] V. Chvátal. *Linear Programming*. Freeman, 1983.

[32] J. Clausen. *Branch and Bound Algorithms – Principles And Examples*. 1999.

[33] C. A. C. Coello. "Evolutionary multi-objective optimization: a historical view of the field". In: *IEEE Computational Intelligence Magazine* 1.1 (Feb. 2006), pp. 28–36. ISSN: 1556-603X. DOI: 10.1109/MCI.2006.1597059.

[34] W. J. Cook. "Solving Traveling Salesman Problems". In: *Algorithms - ESA 2002, 10th Annual European Symposium, Rome, Italy, September 17-21, 2002, Proceedings*. Ed. by R. H. Möhring and R. Raman. Vol. 2461. Lecture Notes in Computer Science. Springer, 2002, p. 1. ISBN: 3-540-44180-8. DOI: 10.1007/3-540-45749-6_1. URL: https://doi.org/10.1007/3-540-45749-6_1.

[35] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. New York, NY, USA: John Wiley & Sons, Inc., 1998. ISBN: 0-471-55894-X.

[36] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press, 2009. ISBN: 0262033844.

[37] D. W. Corne, A. P. Reynolds, and E. Bonabeau. "Swarm Intelligence". In: *Handbook of Natural Computing*. Ed. by G. Rozenberg, T. Bäck, and J. N. Kok. Springer, 2012, pp. 1599–1622. ISBN: 978-3-540-92909-3. DOI: 10.1007/978-3-540-92910-9_48. URL: https://doi.org/10.1007/978-3-540-92910-9_48.

[38] G. A. Croes. "A Method for Solving Traveling-Salesman Problems". In: *Operations Research* 6.6 (1958), pp. 791–812. ISSN: 0030364X, 15265463. URL: http://www.jstor.org/stable/167074.

[39] K. L. Croxton, S. J. García-Dastugue, D. M. Lambert, and D. S. Rogers. "The Supply Chain Management Processes". In: *The International Journal of Logistics Management* 12.2 (2001), pp. 13–36. DOI: 10.1108/09574090110806271. eprint: https://doi.org/10.1108/09574090110806271. URL: https://doi.org/10.1108/09574090110806271.

[40] G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson. "Solution of a Large-Scale Traveling-Salesman Problem". In: *Operations Research* 2.4 (1954), pp. 393–410. DOI: 10.1287/opre.2.4.393. URL: https://doi.org/10.1287/opre.2.4.393.

[41] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. New York, NY, USA: John Wiley & Sons, Inc., 2001. ISBN: 047187339X.

[42] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE Trans. Evolutionary Computation* 6.2 (2002), pp. 182–197. DOI: 10.1109/4235.996017. URL: https://doi.org/10.1109/4235.996017.

[43] M. Dorigo and M. Birattari. "Ant Colony Optimization". In: *Encyclopedia of Machine Learning and Data Mining*. Ed. by C. Sammut and G. I. Webb. Springer, 2017, pp. 56–59. ISBN: 978-1-4899-7685-7. DOI: 10.1007/978-1-4899-7687-1_22. URL: https://doi.org/10.1007/978-1-4899-7687-1_22.

[44] M. Dorigo, V. Maniezzo, and A. Colorni. "Ant system: optimization by a colony of cooperating agents". In: *IEEE Trans. Systems, Man, and Cybernetics, Part B* 26.1 (1996), pp. 29–41. DOI: 10.1109/3477.484436. URL: https://doi.org/10.1109/3477.484436.

[45] M. Dorigo and T. Stützle. *Ant colony optimization*. MIT Press, 2004. ISBN: 978-0-262-04219-2.

[46] B. Drew, A. Plummer, and M. N. Sahinkaya. "A review of wave energy converter technology". In: *Proceedings of the Institution of Mechanical Engineers, Part A: Journal of Power and Energy* 223.8 (2009), pp. 887–902.

[47] A. E. Eiben, T. Bäck, M. Schoenauer, and H. Schwefel, eds. *Parallel Problem Solving from Nature - PPSN V, 5th International Conference, Amsterdam, The Netherlands, September 27-30, 1998, Proceedings*. Vol. 1498. Lecture Notes in Computer Science. Springer, 1998. ISBN: 3-540-65078-4. DOI: 10.1007/BFb0056843. URL: https://doi.org/10.1007/BFb0056843.

[48] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. SpringerVerlag, 2003. ISBN: 3540401849.

[49] M. El Yafrani and B. Ahiod. "Cosolver2B: An efficient local search heuristic for the Travelling Thief Problem". In: *Computer Systems and Applications (AICCSA), 2015 IEEE/ACS 12th International Conference of*. IEEE. 2015, pp. 1–5.

[50] M. El Yafrani and B. Ahiod. "Population-based vs. Single-solution Heuristics for the Travelling Thief Problem". In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. GECCO '16. Denver, Colorado, USA: ACM, 2016, pp. 317–324. ISBN: 978-1-4503-4206-3. DOI: 10.1145/2908812.2908847.

[51] M. El Yafrani and B. Ahiod. "A local search based approach for solving the Travelling Thief Problem: The pros and cons". In: *Applied Soft Computing* 52.Supplement C (2017), pp. 795–804. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2016.09.047.

[52] M. El Yafrani and B. Ahiod. "Efficiently solving the Traveling Thief Problem using hill climbing and simulated annealing". In: *Information Sciences* 432 (2018), pp. 231–244. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2017.12.011.

[53] M. El Yafrani, M. Martins, M. Wagner, B. Ahiod, M. Delgado, and R. Lüders. "A hyperheuristic approach based on low-level heuristics for the travelling thief problem". In: *Genetic Programming and Evolvable Machines* (July 2017). ISSN: 1573-7632. DOI: 10.1007/s10710-017-9308-x. URL: https://doi.org/10.1007/s10710-017-9308-x.

[54] A. F. O. Falcão. "Wave energy utilization: A review of the technologies". In: *Renewable and Sustainable Energy Reviews* 14.3 (2010), pp. 899–918.

[55] J. Falnes. *Ocean waves and oscillating systems: Linear interactions including wave-energy extraction*. Cambridge University Press, 2002. ISBN: 9781139431934.

[56] H. Faulkner, S. Polyakovskiy, T. Schultz, and M. Wagner. "Approximate Approaches to the Traveling Thief Problem". In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. GECCO '15. Madrid, Spain: ACM, 2015, pp. 385–392. ISBN: 978-1-4503-3472-3. DOI: 10.1145/2739480.2754716. URL: http://doi.acm.org/10.1145/2739480.2754716.

[57] C. Fitzgerald and G. Thomas. "A preliminary study on the optimal formation of an array of wave power devices". In: *Proceedings of the 7th European Wave and Tidal Energy Conference*. 2007.

[58] P. B. Garcia-Rosa, G. Bacelli, and J. V. Ringwood. "Control-informed optimal array layout for wave farms". In: *IEEE Transactions on Sustainable Energy* 6.2 (2015), pp. 575–582.

[59] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. ISBN: 0-7167-1044-7.

[60] M. R. Garey, D. S. Johnson, and R. Sethi. "The Complexity of Flowshop and Jobshop Scheduling". In: *Math. Oper. Res.* 1.2 (1976), pp. 117–129. DOI: 10.1287/moor.1.2.117. URL: https://doi.org/10.1287/moor.1.2.117.

[61] M. Gendreau and J.-Y. Potvin. *Handbook of Metaheuristics*. 2nd. Springer Publishing Company, Incorporated, 2010. ISBN: 1441916636.

[62] I. P. Gent and T. Walsh. "CSPlib: A Benchmark Library for Constraints". In: *Principles and Practice of Constraint Programming – CP'99: 5th International Conference, CP'99, Alexandria, VA, USA, October 11-14, 1999. Proceedings*. Ed. by J. Jaffar. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 480–481. ISBN: 978-3-540-48085-3. DOI: 10.1007/978-3-540-48085-3_36. URL: https://doi.org/10.1007/978-3-540-48085-3_36.

[63] F. W. Glover and G. A. Kochenberger. *Handbook of Metaheuristics*. Hardcover. Jan. 2003.

[64] F. Glover and M. Laguna. *TABU search*. Kluwer, 1999. ISBN: 978-0-7923-9965-0.

[65] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. New York: Addison-Wesley, 1989.

[66] GOODYEAR. *Factors Affecting Truck Fuel Economy*. Akron, OH, USA, Nov. 2011. URL: https://www.goodyeartrucktires.com/pdf/resources/publications/factors_affecting_truck_fuel_economy.pdf.

[67] N. I. Gould, D. Orban, and P. L. Toint. "CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization". In: *Computational Optimization and Applications* 60.3 (2015), pp. 545–557.

[68] R. L. Graham and P. Hell. "On the History of the Minimum Spanning Tree Problem". In: *IEEE Ann. Hist. Comput.* 7.1 (Jan. 1985), pp. 43–57. ISSN: 1058-6180. DOI: 10.1109/MAHC.1985.10011. URL: https://doi.org/10.1109/MAHC.1985.10011.

[69] B. C. Gupta and V. P. Prakash. "Greedy heuristics for the Travelling Thief Problem". In: *2015 39th National Systems Conference (NSC)*. Dec. 2015, pp. 1–5. DOI: 10.1109/NATSYS.2015.7489116.

[70] J. Hals, J. Falnes, and T. Moan. "A comparison of selected strategies for adaptive control of wave energy converters". In: *Journal of Offshore Mechanics and Arctic Engineering* 133.3 (2011), p. 031101.

[71] N. Hansen and A. Ostermeier. "Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation". In: *Proceedings of IEEE International Conference on Evolutionary Computation*. 1996, pp. 312–317.

[72] M. Held and R. M. Karp. "A Dynamic Programming Approach to Sequencing Problems". In: *Proceedings of the 1961 16th ACM National Meeting*. ACM '61. ACM, 1961, pp. 71.201–71.204.

[73] K. Helsgaun. "An effective implementation of the Lin-Kernighan traveling salesman heuristic". In: *European Journal of Operational Research* 126.1 (2000), pp. 106–130. DOI: 10.1016/S0377-2217(99)00284-2. URL: https://doi.org/10.1016/S0377-2217(99)00284-2.

[74] M. A. Hemer and D. A. Griffin. "The wave energy resource along Australia's southern margin". In: *Renewable and Sustainable Energy* 2.4, 043108 (2010). DOI: http://dx.doi.org/10.1063/1.3464753. URL: http://scitation.aip.org/content/aip/journal/jrse/2/4/10.1063/1.3464753.

[75] "Approximation Algorithms for NP-Hard Problems". In: *SIGACT News* 28.2 (June 1997). Ed. by D. S. Hochba, pp. 40–52. ISSN: 0163-5700. DOI: 10.1145/261342.571216. URL: http://doi.acm.org/10.1145/261342.571216.

[76] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992. ISBN: 0262082136.

[77] J. Hooker. *Logic-based methods for optimization: combining optimization and constraint satisfaction*. Vol. 2. John Wiley & Sons, 2011.

[78] J. N. Hooker. "Logic, optimization, and constraint programming". In: *INFORMS Journal on Computing* 14.4 (2002), pp. 295–321.

[79] H. H. Hoos and T. Stützle. "SATLIB: An Online Resource for Research on SAT". In: IOS Press, 2000, pp. 283–292.

[80] H. H. Hoos and T. Stützle. "Stochastic Local Search Algorithms: An Overview". In: *Springer Handbook of Computational Intelligence*. Ed. by J. Kacprzyk and W. Pedrycz. Springer, 2015, pp. 1085–1105. ISBN: 978-3-662-43504-5. DOI: 10.1007/978-3-662-43505-2_54. URL: https://doi.org/10.1007/978-3-662-43505-2_54.

[81] D. Hoy and E. Nikolova. "Approximately Optimal Risk-Averse Routing Policies via Adaptive Discretization". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*. Ed. by B. Bonet and S. Koenig. AAAI Press, 2015, pp. 3533–3539. ISBN: 978-1-57735-698-1.

[82] C.-L. Hwang and A. S. M. Masud. *Multiple objective decision making—methods and applications: a state-of-the-art survey*. Vol. 164. Springer Science & Business Media, 2012.

[83] J. E. D. Jr. and R. B. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*. Prentice Hall series in computational mathematics. Prentice Hall, 1983. ISBN: 978-0-13-627216-8.

[84] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, Berlin, Germany, 2004.

[85] J. Kennedy and R. Eberhart. "Particle swarm optimization". In: *Neural Networks, 1995. Proceedings., IEEE International Conference on*. Vol. 4. Nov. 1995, 1942–1948 vol.4. DOI: 10.1109/ICNN.1995.488968.

[86] S. Kirkpatrick, D. G. Jr., and M. P. Vecchi. "Optimization by Simmulated Annealing". In: *Science* 220.4598 (1983), pp. 671–680.

[87] K. Klamroth, S. Mostaghim, B. Naujoks, S. Poles, R. Purshouse, G. Rudolph, S. Ruzika, S. Sayın, M. M. Wiecek, and X. Yao. "Multiobjective optimization for interwoven systems". In: *Journal of Multi-Criteria Decision Analysis* 24.1-2 (2017). mcda.1598, pp. 71–81. ISSN: 1099-1360. DOI: 10.1002/mcda.1598. URL: http://dx.doi.org/10.1002/mcda.1598.

[88] R. Kolisch and A. Sprecher. "PSPLIB - A project scheduling problem library: OR Software - ORSEP Operations Research Software Exchange Program". In: *European Journal of Operational Research* 96.1 (1997), pp. 205–216. ISSN: 0377-2217. DOI: https://doi.org/10.1016/S0377-2217(96)00170-1. URL: http://www.sciencedirect.com/science/article/pii/S0377221796001701.

[89] T. Kötzing, F. Neumann, H. Röglin, and C. Witt. "Theoretical Properties of Two ACO Approaches for the Traveling Salesman Problem". In: *Swarm Intelligence*. Ed. by M. Dorigo, M. Birattari, G. A. Di Caro, R. Doursat, A. P. Engelbrecht, D. Floreano, L. M. Gambardella, R. Groß, E. Şahin, H. Sayama, and T. Stützle. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 324–335. ISBN: 978-3-642-15461-4.

[90] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992. ISBN: 0-262-11170-5.

[91] M. Lagoun, A. Benalia, and M. Benbouzid. "Ocean wave converters: State of the art and current status". In: *IEEE International Energy Conference*. 2010, pp. 636–641.

[92] D. M. Lambert, M. C. Cooper, and J. D. Pagh. "Supply chain management: implementation issues and research opportunities". In: *The international journal of logistics management* 9.2 (1998), pp. 1–20.

[93] O. E. Landman**. "The inheritance of acquired characteristics". In: *Annual Review of Genetics* 25.1 (1991), pp. 1–20.

[94] W. B. Langdon, R. Poli, N. F. McPhee, and J. R. Koza. "Genetic Programming: An Introduction and Tutorial, with a Survey of Techniques and Applications". In: *Computational Intelligence: A Compendium*. Ed. by J. Fulcher and L. C. Jain. Vol. 115. Studies in Computational Intelligence. Springer, 2008, pp. 927–1028. ISBN: 978-3-540-78292-6. DOI: 10.1007/978-3-540-78293-3_22. URL: https://doi.org/10.1007/978-3-540-78293-3_22.

[95] G. Laporte. "The Traveling Salesman Problem, the Vehicle Routing Problem, and Their Impact on Combinatorial Optimization". In: *IJSDS* 1.2 (2010), pp. 82–92. DOI: 10.4018/jsds.2010040104. URL: https://doi.org/10.4018/jsds.2010040104.

[96] P. Larrañaga, C. Kuijpers, R. Murga, I. Inza, and S. Dizdarevic. "Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators". In: *Artificial Intelligence Review* 13.2 (Apr. 1999), pp. 129–170. ISSN: 1573-7462. DOI: 10.1023/A:1006529012972. URL: https://doi.org/10.1023/A:1006529012972.

[97] C.-H. Lee. *WAMIT Theory Manual*. Massachusetts Institute of Technology, 1995.

[98] J. van Leeuwen, ed. *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*. Elsevier and MIT Press, 1990. ISBN: 0-444-88071-2.

[99] X. Li, K. Tang, M. N. Omidvar, Z. Yang, and K. Qin. *Benchmark Functions for the CEC'2013 Special Session and Competition on Large-Scale Global Optimization*. 2013.

[100] C. Lin, K. Choy, G. Ho, S. Chung, and H. Lam. "Survey of Green Vehicle Routing Problem: Past and future trends". In: *Expert Systems with Applications* 41.4, Part 1 (2014), pp. 1118–1138. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2013.07.107.

[101] S. Lin and B. W. Kernighan. "An Effective Heuristic Algorithm for the Traveling-Salesman Problem". In: *Operations Research* 21.2 (1973), pp. 498–516. DOI: 10.1287/opre.21.2.498. URL: https://doi.org/10.1287/opre.21.2.498.

[102] C. Linton. "Radiation and diffraction of water waves by a submerged sphere in finite depth". In: *Ocean Engineering* 18.1 (1991), pp. 61–74.

[103] I. López, J. Andreu, S. Ceballos, I. Martínez de Alegría, and I. Kortabarria. "Review of wave energy technologies and the necessary power-equipment". In: *Renewable and Sustainable Energy Reviews* 27 (2013), pp. 413–434.

[104] N. Lourenço, F. B. Pereira, and E. Costa. "An Evolutionary Approach to the Full Optimization of the Traveling Thief Problem". In: *Evolutionary Computation in Combinatorial Optimization: 16th European Conference, EvoCOP 2016, Porto, Portugal, March 30 – April 1, 2016, Proceedings*. Ed. by F. Chicano, B. Hu, and P. García-Sánchez. Cham: Springer International Publishing, 2016, pp. 34–45. ISBN: 978-3-319-30698-8. DOI: 10.1007/978-3-319-30698-8_3. URL: https://doi.org/10.1007/978-3-319-30698-8_3.

[105] P. A. Lynn. *Electricity from Wave and Tide: An Introduction to Marine Energy*. John Wiley & Sons, 2013.

[106] L. D. Mann. "Application of ocean observations & analysis: The CETO wave energy project". In: *Operational Oceanography in the 21st Century*. Springer, 2011, pp. 721–729.

[107] L. Mann, A. Burns, and M. Ottaviano. "CETO, a carbon free wave power energy provider of the future". In: *Proceedings of the 7th European Wave and Tidal Energy Conference*. 2007.

[108] S. Martello, D. Pisinger, and P. Toth. "Dynamic Programming and Strong Bounds for the 0-1 Knapsack Problem". In: *Manage. Sci.* 45.3 (Mar. 1999), pp. 414–424. ISSN: 0025-1909. DOI: 10.1287/mnsc.45.3.414. URL: http://dx.doi.org/10.1287/mnsc.45.3.414.

[109] S. Martello and P. Toth. "A Note on 0.5-Bounded Greedy Algorithms for the 0/1 Knapsack Problem". In: *Inf. Process. Lett.* 44.4 (1992), pp. 221–222. DOI: 10.1016/0020-0190(92)90089-E. URL: https://doi.org/10.1016/0020-0190(92)90089-E.

[110] S. Martello and P. Toth. "An Exact Algorithm for the Two-Constraint 0 - 1 Knapsack Problem". In: *Operations Research* 51.5 (2003), pp. 826–835. DOI: 10.1287/opre.51.5.826.16757. URL: https://doi.org/10.1287/opre.51.5.826.16757.

[111] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990. ISBN: 0-471-92420-2.

[112] S. Martello and P. Toth. "Upper Bounds and Algorithms for Hard 0-1 Knapsack Problems". In: *Operations Research* 45.5 (1997), pp. 768–778. DOI: 10.1287/opre.45.5.768. URL: https://doi.org/10.1287/opre.45.5.768.

[113] M. S. R. Martins, M. El Yafrani, M. R. B. S. Delgado, M. Wagner, B. Ahiod, and R. Lüders. "HSEDA: A Heuristic Selection Approach Based on Estimation of Distribution Algorithm for the Travelling Thief Problem". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '17. Berlin, Germany: ACM, 2017, pp. 361–368. ISBN: 978-1-4503-4920-8. DOI: 10.1145/3071178.3071235. URL: http://doi.acm.org/10.1145/3071178.3071235.

[114] B. Mayoh, E. Tyugu, and J. Penjam. *Constraint programming*. Vol. 131. Springer Science & Business Media, 2013.

[115] A. McCabe, G. Aggidis, and M. Widden. "Optimizing the shape of a surge-and-pitch wave energy collector using a genetic algorithm". In: *Renewable Energy* 35.12 (2010), pp. 2767–2775.

[116] Y. Mei, X. Li, F. Salim, and X. Yao. "Heuristic evolution with Genetic Programming for Traveling Thief Problem". In: *IEEE Congress on Evolutionary Computation, CEC 2015, Sendai, Japan, May 25-28, 2015*. IEEE, 2015, pp. 2753–2760. ISBN: 978-1-4799-7492-4. DOI: 10.1109/CEC.2015.7257230. URL: https://doi.org/10.1109/CEC.2015.7257230.

[117] Y. Mei, X. Li, and X. Yao. "Improving Efficiency of Heuristics for the Large Scale Traveling Thief Problem". In: *Simulated Evolution and Learning: 10th International Conference, SEAL 2014, Dunedin, New Zealand, December 15-18, 2014. Proceedings*. Cham: Springer International Publishing, 2014, pp. 631–643. ISBN: 978-3-319-13563-2. DOI: 10.1007/978-3-319-13563-2_53.

[118] Y. Mei, X. Li, and X. Yao. "On investigation of interdependence between sub-problems of the Travelling Thief Problem". In: *Soft Comput.* 20.1 (2016), pp. 157–172. DOI: 10.1007/s00500-014-1487-2. URL: https://doi.org/10.1007/s00500-014-1487-2.

[119] Z. Michalewicz. *How to Solve It: Modern Heuristics 2e*. Berlin, Heidelberg: Springer-Verlag, 2010. ISBN: 3642061346.

[120] Z. Michalewicz. "Quo Vadis, Evolutionary Computation? - On a Growing Gap between Theory and Practice". In: *Advances in Computational Intelligence - IEEE World Congress on Computational Intelligence, WCCI 2012, Brisbane, Australia, June 10-15, 2012. Plenary/Invited Lectures*. Ed. by J. Liu, C. Alippi, B. Bouchon-Meunier, G. W. Greenwood, and H. A. Abbass. Vol. 7311. Lecture Notes in Computer Science. Springer, 2012, pp. 98–121. ISBN: 978-3-642-30686-0. DOI: 10.1007/978-3-642-30687-7_6. URL: https://doi.org/10.1007/978-3-642-30687-7_6.

[121] B. L. Miller and D. E. Goldberg. "Genetic Algorithms, Tournament Selection, and the Effects of Noise". In: *Complex Systems* 9.3 (1995). URL: http://www.complex-systems.com/abstracts/v09_i03_a02.html.

[122] M. Mitchell. *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998. ISBN: 0262631857.

[123] M. Mitchell. *An introduction to genetic algorithms*. MIT Press, 1998. ISBN: 978-0-262-63185-3.

[124] M. Mohamed, G. Janiga, E. Pap, and D. Thévenin. "Multi-objective optimization of the airfoil shape of Wells turbine used for wave energy conversion". In: *Energy* 36.1 (2011), pp. 438–446.

[125] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell. "Branch-and-bound Algorithms". In: *Discret. Optim.* 19.C (Feb. 2016), pp. 79–102. ISSN: 1572-5286. DOI: 10.1016/j.disopt.2016.01.005. URL: http://dx.doi.org/10.1016/j.disopt.2016.01.005.

[126] P. Moscato et al. "On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms". In: *Caltech concurrent computation program, C3P Report* 826 (1989), p. 1989.

[127] F. Neri, C. Cotta, and P. Moscato, eds. *Handbook of Memetic Algorithms*. Vol. 379. Studies in Computational Intelligence. Springer, 2012. ISBN: 978-3-642-23246-6. DOI: 10.1007/978-3-642-23247-3. URL: https://doi.org/10.1007/978-3-642-23247-3.

[128] F. Neumann and A. Q. Nguyen. "On the Impact of Utility Functions in Interactive Evolutionary Multi-objective Optimization". In: *Simulated Evolution and Learning - 10th International Conference, SEAL 2014, Dunedin, New Zealand, December 15-18, 2014. Proceedings*. Ed. by G. Dick, W. N. Browne, P. A. Whigham, M. Zhang, L. T. Bui, H. Ishibuchi, Y. Jin, X. Li, Y. Shi, P. Singh, K. C. Tan, and K. Tang. Vol. 8886. Lecture Notes in Computer Science. Springer, 2014, pp. 419–430. ISBN: 978-3-319-13562-5. DOI: 10.1007/978-3-319-13563-2_36. URL: http://dx.doi.org/10.1007/978-3-319-13563-2_36.

[129] F. Neumann, S. Polyakovskiy, M. Skutella, L. Stougie, and J. Wu. "A Fully Polynomial Time Approximation Scheme for Packing While Traveling". In: *CoRR* abs/1702.05217 (2017). arXiv: 1702.05217. URL: http://arxiv.org/abs/1702.05217.

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

[131] J. Nocedal and S. J. Wright. *Numerical Optimization*. second. New York, NY, USA: Springer, 2006.

[132] G. Nunes, D. Valério, P. Beirao, and J. S. Da Costa. "Modelling and control of a wave energy converter". In: *Renewable Energy* 36.7 (2011), pp. 1913–1921.

[133] M. N. Omidvar, X. Li, Y. Mei, and X. Yao. "Cooperative Co-Evolution With Differential Grouping for Large Scale Optimization". In: *IEEE Trans. Evolutionary Computation* 18.3 (2014), pp. 378–393. DOI: 10.1109/TEVC.2013.2281543. URL: https://doi.org/10.1109/TEVC.2013.2281543.

[134] M. N. Omidvar, X. Li, and K. Tang. "Designing benchmark problems for large-scale continuous optimization". In: *Inf. Sci.* 316 (2015), pp. 419–436. DOI: 10.1016/j.ins.2014.12.062. URL: https://doi.org/10.1016/j.ins.2014.12.062.

[135] E. C. Osuna, W. Gao, F. Neumann, and D. Sudholt. "Speeding up evolutionary multi-objective optimisation through diversity-based parent selection". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, Berlin, Germany, July 15-19, 2017*. Ed. by P. A. N. Bosman. ACM, 2017, pp. 553–560. ISBN: 978-1-4503-4920-8. DOI: 10.1145/3071178.3080294. URL: http://doi.acm.org/10.1145/3071178.3080294.

[136] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1982. ISBN: 0-13-152462-3.

[137] D. Pisinger. *Advanced Generator for 0-1 Knapsack Problem*. URL: http://www.diku.dk/~pisinger/codes.html.

[138] D. Pisinger. "Where Are the Hard Knapsack Problems?" In: *Comput. Oper. Res.* 32.9 (Sept. 2005), pp. 2271–2284. ISSN: 0305-0548. DOI: 10.1016/j.cor.2004.03.002.

[139] S. Polyakovskiy, M. R. Bonyadi, M. Wagner, Z. Michalewicz, and F. Neumann. "A comprehensive benchmark set and heuristics for the traveling thief problem". In: *Genetic and Evolutionary Computation Conference, GECCO '14, Vancouver, BC, Canada, July 12-16, 2014*. Ed. by D. V. Arnold. ACM, 2014, pp. 477–484. ISBN: 978-1-4503-2662-9. DOI: 10.1145/2576768.2598249. URL: http://doi.acm.org/10.1145/2576768.2598249.

[140] S. Polyakovskiy and F. Neumann. "The Packing While Traveling Problem". In: *European Journal of Operational Research* 258.2 (2017), pp. 424–439. DOI: 10.1016/j.ejor.2016.09.035. URL: https://doi.org/10.1016/j.ejor.2016.09.035.

[141] M. A. Potter and K. A. De Jong. "A cooperative coevolutionary approach to function optimization". In: *Parallel Problem Solving from Nature — PPSN III*. Ed. by Y. Davidor, H.-P. Schwefel, and R. Männer. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 249–257. ISBN: 978-3-540-49001-2.

[142] C. R. Reeves and J. E. Rowe. *Genetic Algorithms: Principles and Perspectives: A Guide to GA Theory*. Norwell, MA, USA: Kluwer Academic Publishers, 2002. ISBN: 1402072406.

[143] P. Refalo. "Principles and Practice of Constraint Programming – CP 2004". In: Springer, 2004. Chap. Impact-Based Search Strategies for Constraint Programming, pp. 557–571.

[144] C. Rego, D. Gamboa, F. Glover, and C. Osterman. "Traveling salesman problem heuristics: Leading methods, implementations and latest advances". In: *European Journal of Operational Research* 211.3 (2011), pp. 427–441. DOI: 10.1016/j.ejor.2010.09.010. URL: https://doi.org/10.1016/j.ejor.2010.09.010.

[145] G. Reinelt. "TSPLIB- a Traveling Salesman Problem Library". In: *ORSA Journal of Computing* 3.4 (1991), pp. 376–384.

[146] F. Rossi, P. v. Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. New York, NY, USA: Elsevier Science Inc., 2006. ISBN: 0444527265.

[147] F. Rothlauf. *Design of Modern Heuristics: Principles and Application*. 1st. Springer Publishing Company, Incorporated, 2011. ISBN: 3540729615.

[148] G. Rozenberg, T. Bäck, and J. N. Kok, eds. *Handbook of Natural Computing*. Springer, 2012. ISBN: 978-3-540-92909-3. DOI: 10.1007/978-3-540-92910-9. URL: https://doi.org/10.1007/978-3-540-92910-9.

[149] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. 2nd ed. Pearson Education, 2003. ISBN: 0137903952.

[150] S. H. Salter. "Wave power". In: *Nature* 249.5459 (1974), pp. 720–724.

[151] A. Schrijver. *Theory of Linear and Integer Programming*. New York, NY, USA: John Wiley & Sons, Inc., 1986. ISBN: 0-471-90854-1.

[152] J. T. Scruggs, S. M. Lattanzio, A. A. Taflanidis, and I. L. Cassidy. "Optimal causal control of a wave energy converter in a random sea". In: *Applied Ocean Research* 42.2013 (2013), pp. 1–15. DOI: 10.1016/j.apor.2013.03.004.

[153] N. Y. Sergiienko, B. S. Cazzolato, B. Ding, and M. Arjomandi. *Frequency domain model of the three-tether WECs array*. [Online; accessed 2-February-2016]. 2016. DOI: http://dx.doi.org/10.13140/RG.2.1.1917.0324. URL: https://www.researchgate.net/publication/291972368_Frequency_domain_model_of_the_three-tether_WECs_array.

[154] E. A. Silver. "An Overview of Heuristic Solution Methods". In: *IN PROCEEDINGS OF THE 7TH ANNUAL INTERNATIONAL CONFERENCE ON INDUSTRIAL ENGINEERING THEORY, APPLICATIONS AND PRACTICE*. 2002.

[155] A. Sinha, P. Malo, and K. Deb. "A Review on Bilevel Optimization: From Classical to Evolutionary Approaches and Applications". In: *IEEE Trans. Evolutionary Computation* 22.2 (2018), pp. 276–295. DOI: 10.1109/TEVC.2017.2712906. URL: https://doi.org/10.1109/TEVC.2017.2712906.

[156] M. Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997. ISBN: 978-0-534-94728-6.

[157] A. Sosa-Ascencio, G. Ochoa, H. Terashima-Marin, and S. E. Conant-Pablos. "Grammar-based generation of variable-selection heuristics for constraint satisfaction problems". In: *Genetic Programming and Evolvable Machines* 17.2 (June 2016), pp. 119–144. ISSN: 1573-7632. DOI: 10.1007/s10710-015-9249-1. URL: https://doi.org/10.1007/s10710-015-9249-1.

[158] M. A. Srokosz. "The submerged sphere as an absorber of wave power". In: *Fluid Mechanics* 95.4 (1979), pp. 717–741.

[159] J. Stolk, I. Mann, A. Mohais, and Z. Michalewicz. "Combining vehicle routing and packing for optimal delivery schedules of water tanks". In: *OR Insight* 26.3 (2013), pp. 167–190. DOI: 10.1057/ori.2013.1. URL: http://dx.doi.org/10.1057/ori.2013.1.

[160] A. Strzeźek, L. Trammer, and M. Sydow. "DiverGene: Experiments on controlling population diversity in genetic algorithm with a dispersion operator". In: *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)*. Sept. 2015, pp. 155–162. DOI: 10.15439/2015F411.

[161] T. Stützle and H. H. Hoos. "MAX MIN Ant System". In: *Future Generation Computer Systems* 16.8 (2000), pp. 889–914.

[162] H. A. Taha. *Operations Research: An Introduction (9th Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2010. ISBN: 013255593x.

[163] K. Tang, X. Li, P. N. Suganthan, Z. Yang, and T. Weise. *Benchmark functions for the cec'2010 special session and competition on large-scale global optimization*. Tech. rep. Nature Inspired Computation and Applications Laboratory, 2009.

[164] G. Tao and Z. Michalewicz. "Inver-over Operator for the TSP". In: *Parallel Problem Solving from Nature - PPSN V, 5th International Conference, Amsterdam, The Netherlands, September 27-30, 1998, Proceedings*. Ed. by A. E. Eiben, T. Bäck, M. Schoenauer, and H. Schwefel. Vol. 1498. Lecture Notes in Computer Science. Springer, 1998, pp. 803–812. ISBN: 3-540-65078-4. DOI: 10.1007/BFb0056922. URL: https://doi.org/10.1007/BFb0056922.

[165] P. Toth. "Dynamic programming algorithms for the Zero-One Knapsack Problem". In: *Computing* 25.1 (1980), pp. 29–45. DOI: 10.1007/BF02243880. URL: https://doi.org/10.1007/BF02243880.

[166] E. P. K. Tsang. *Foundations of constraint satisfaction*. Computation in cognitive science. Academic Press, 1993. ISBN: 978-0-12-701610-8.

[167] L. Vanneschi and R. Poli. "Genetic Programming - Introduction, Applications, Theory and Open Issues". In: *Handbook of Natural Computing*. Ed. by G. Rozenberg, T. Bäck, and J. N. Kok. Springer, 2012, pp. 709–739. ISBN: 978-3-540-92909-3. DOI: 10.1007/978-3-540-92910-9_24. URL: https://doi.org/10.1007/978-3-540-92910-9_24.

[168] M. Wagner. "Stealing Items More Efficiently with Ants: A Swarm Intelligence Approach to the Travelling Thief Problem". In: *Swarm Intelligence: 10th International Conference, ANTS 2016, Brussels, Belgium, September 7-9, 2016, Proceedings*. Ed. by M. Dorigo, M. Birattari, X. Li, M. López-Ibáñez, K. Ohkura, C. Pinciroli, and T. Stützle. Springer, 2016, pp. 273–281.

[169] M. Wagner, M. Lindauer, M. Mısır, S. Nallaperuma, and F. Hutter. "A case study of algorithm selection for the traveling thief problem". In: *Journal of Heuristics* (2017), pp. 1–26.

[170] B. L. Welch. "The Generalization of 'Student's' Problem when Several Different Population Variances are Involved". In: *Biometrika* 34.1/2 (1947), pp. 28–35. ISSN: 00063444. URL: http://www.jstor.org/stable/2332510.

[171] D. Wolpert and W. G. Macready. "No free lunch theorems for optimization". In: *IEEE Trans. Evolutionary Computation* 1.1 (1997), pp. 67–82. DOI: 10.1109/4235.585893. URL: https://doi.org/10.1109/4235.585893.

[172] G. X. Wu. "Radiation and Diffraction by a Submerged Sphere Advancing in Water Waves of Finite Depth". In: *Proceedings: Mathematical and Physical Sciences* 448.1932 (1995), pp. 29–54. DOI: 10.2307/52502.

[173] J. Wu, S. Polyakovskiy, and F. Neumann. "On the Impact of the Renting Rate for the Unconstrained Nonlinear Knapsack Problem". In: *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016*. 2016, pp. 413–419.

[174] J. Wu, S. Polyakovskiy, M. Wagner, and F. Neumann. "Evolutionary Computation plus Dynamic Programming for the Bi-Objective Travelling Thief Problem". In: *CoRR* abs/1802.02434 (2018). arXiv: 1802.02434. URL: http://arxiv.org/abs/1802.02434.

[175] J. Wu, S. Shekh, N. Y. Sergiienko, B. S. Cazzolato, B. Ding, F. Neumann, and M. Wagner. "Fast and Effective Optimisation of Arrays of Submerged Wave Energy Converters". In: *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016*. 2016, pp. 1045–1052.

[176] J. Wu, M. Wagner, S. Polyakovskiy, and F. Neumann. "Exact Approaches for the Travelling Thief Problem". In: *Simulated Evolution and Learning - 11th International Conference, SEAL 2017, Shenzhen, China, November 10-13, 2017, Proceedings*. 2017, pp. 110–121.

[177] J. Xie, Y. Mei, A. T. Ernst, X. Li, and A. Song. "A Bi-Level Optimization Model for Grouping Constrained Storage Location Assignment Problems". In: *IEEE Trans. Cybernetics* 48.1 (2018), pp. 385–398. DOI: 10.1109/TCYB.2016.2638820. URL: https://doi.org/10.1109/TCYB.2016.2638820.

[178] M. E. Yafrani, S. Chand, A. Neumann, B. Ahiod, and M. Wagner. "Multi-objectiveness in the Single-objective Traveling Thief Problem". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '17. Berlin, Germany: ACM, 2017, pp. 107–108. ISBN: 978-1-4503-4939-0. DOI: 10.1145/3067695.3076010. URL: http://doi.acm.org/10.1145/3067695.3076010.

[179] G. Yang and E. Nikolova. "Approximation Algorithms for Route Planning with Nonlinear Objectives". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*. Ed. by D. Schuurmans and M. P. Wellman. AAAI Press, 2016, pp. 3209–3217.

[180] Y. Zhou and J. He. "A Runtime Analysis of Evolutionary Algorithms for Constrained Optimization Problems". In: *IEEE Trans. Evolutionary Computation* 11.5 (2007), pp. 608–619. DOI: 10.1109/TEVC.2006.888929. URL: http://dx.doi.org/10.1109/TEVC.2006.888929.

[181] E. Zitzler, K. Deb, and L. Thiele. "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results". In: *Evolutionary Computation* 8.2 (2000), pp. 173–195. DOI: 10.1162/106365600568202. URL: https://doi.org/10.1162/106365600568202.

[182] E. Zitzler and S. Künzli. "Indicator-Based Selection in Multiobjective Search". In: *Parallel Problem Solving from Nature - PPSN VIII, 8th International Conference, Birmingham, UK, September 18-22, 2004, Proceedings*. Ed. by X. Yao, E. K. Burke, J. A. Lozano, J. Smith, J. J. M. Guervós, J. A. Bullinaria, J. E. Rowe, P. Tiňo, A. Kabán, and H. Schwefel. Vol. 3242. Lecture Notes in Computer Science. Springer, 2004, pp. 832–842. ISBN: 3-540-23092-0. DOI: 10.1007/978-3-540-30217-9_84. URL: https://doi.org/10.1007/978-3-540-30217-9_84.

[183] E. Zitzler, M. Laumanns, and L. Thiele. "SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization". In: (2002), pp. 95–100.

[184] E. Zitzler and L. Thiele. "Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study". In: *Parallel Problem Solving from Nature - PPSN V, 5th International Conference, Amsterdam, The Netherlands, September 27-30, 1998, Proceedings*. Ed. by A. E. Eiben, T. Bäck, M. Schoenauer, and H. Schwefel. Vol. 1498. Lecture Notes in Computer Science. Springer, 1998, pp. 292–304. ISBN: 3-540-65078-4. DOI: 10.1007/BFb0056872. URL: https://doi.org/10.1007/BFb0056872.