UNIVERSITY OF MACEDONIA

GRADUATE PROGRAM

DEPARTMENT OF APPLIED INFORMATICS

A REDUCED VARIABLE NEIGHBORHOOD SEARCH APPROACH FOR THE TRAVELING THIEF PROBLEM

Master Thesis

of

Fotoglou Ioakeim

Thessaloniki, October 30, 2019

A REDUCED VARIABLE NEIGHBORHOOD SEARCH APPROACH FOR THE TRAVELING THIEF PROBLEM

Fotoglou Ioakeim

Bachelor in Mathematics, University of Crete, 2015

Master Thesis

submitted in the partial fulfillment of the demands for the

MASTER OF SCIENCE IN APPLIED INFORMATICS

Supervising Professor
Sifaleras Angelo

Approved by the three-member committee 30/10/2019

Sifaleras Angelo	Samaras Nikolaos	Refanidis Ioannis
.....	.....	.....

Fotoglou Ioakeim

.....

Abstract

While single objective optimization problems have been extensively studied in literature, their narrow scope and very specific field of focus remain a major limiting factor in pragmatic and practical applications. Problems encountered in real-world scenarios are usually a mixture of interdependent and co-relational factors that have dynamic implications in the solution process. In this thesis we examine one such complex problem, the Travelling Thief Problem, derived from the interweaving of two component subproblems; namely the TSP and Knapsack problems, into a coherent interdependent combinatorial optimization problem through the use of the RVNS and VNS metaheuristic methods.

Keywords: Reduced Variable Neighborhood Search, Variable Neighborhood Search, GRASP, Traveling Thief Problem, Travelling Salesman Problem, Knapsack Problem, Metaheuristic Method.

THANKS

I would like to thank my supervising professor for aiding significantly in the completion of this thesis through his focused and resolute guidance. His cooperation and experience have been invaluable in the compilation of this work.

I would also like to thank my family, whose aid, help and support have allowed me to pursue my academic interests since an early age. None of this would have been possible without them.

This work is dedicated to those researchers who have faced setbacks, sleepless nights, difficult decisions and frustrating days to pursue their passion.

CONTENTS

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Motivations

Combinatorial optimization is a novel and under-researched field with numerous real-world applications that have gathered significant research interest over the last few years. The practical importance of such problems in a wide variety of fields including, but not limited to, industry, transport, logistics, scheduling etc. have the potential to produce significant contributions in every one of those fields.

This research was motivated by an active interest in investigating how heuristic methods can effectively contribute to the realization of rapid and efficient solutions to problems that are difficult and very resource demanding, by investigating the production of good and efficient solutions under the prism of VNS.

1.2 Context of the study

This thesis was authored in partial fulfilment of the requirements for the Master in Applied Informatics of the University of Macedonia graduate program. The code for all heuristic methods utilized in this research was authored in Python and run locally without parallelization considerations. The datasets used were provided by the inventors of the Travelling Thief problem in the University of Adelaide in Australia and no modifications whatsoever have taken place in order to allow future research to exploit the outcome of this research to hopefully further the insight into advancing heuristic methods for the solution of combinatorial optimization problems.

1.3 Objectives and contribution

The present study has been undertaken in order to enhance the researcher's understanding of hard combinatorial optimization problems and with the hope of producing a worthwhile contribution to other researchers and students in the fields of heuristics and hard problem solving. Some of the most up to date research is utilised and select components from the datasets of the inventors of TTP are utilized to craft an RVNS approach to drawing efficient and rapid solutions for these complex problems.

1.4 Overview of the thesis

This thesis is organised in seven chapters, *Chapter 1* presents the introduction to the thesis, the motivation of the researcher and some organizing principles behind this document.

Chapter 2 dwells into the theoretical background, presenting a building-block approach to the Travelling Thief Problem which is the main problem examined in this thesis by elaborating on how it evolved through a process of natural evolution through its component subproblems. Subsequently some of the most prevalent metaheuristic methods that are used to tackle such hard problems are being introduced in short form with some of their prevalent characteristics being enumerated in a brief manner.

VNS which is the main heuristic method utilized in the experimental and theoretical phase of the present study is examined in more detail in its own subsection and the differences between it and RVNS are presented in the form of pseudocode as well as in a theoretical sense, drawing upon the relevant literature. A limited number of approaches to the construction function heuristics is presented including the three main ones utilized in constructing the initial solutions of this study, i.e. a random heuristic, a greedy/hill climbing approach and a more advanced probabilistic greedy heuristic based on the construction function of the GRASP heuristic which is the one that, as will be detailed in the relevant section, produces the most efficient initial solutions. Closing this subsection some methods for multi-objective optimization, namely the Weighted Sum method and the ϵ-constraint method are presented.

Chapter 3 begins by introducing the Travelling Thief Problem, the inspiration behind its creation and how the motivation of introducing an interdependency factor between difficult problems has led to a melding of components from both constituent problems into

a unified problem with a combined dataset whose solution is the result of contributions from factors of both problems.

The inderdependency is further examined in its own special subsection where the critical importance of understanding and unifying the solutions to the Travelling Salesman Problem and the Knapsack Problem are being investigated through the prism of relevant research literature, drawing upon the significant body of research suggesting that solutions to co-related problems should not be examined on a per-problem base but rather as a unified and dynamic problem that constantly changes. Similar arguments which are found in the bibliography for the Vehicle Routing Problem, which is also a combinatorial optimization problem facing the same challenges and demands, are also presented to further enhance the author's position.

Subsequently, the chapter concludes by examining in great detail some of the main Metaheuristic approaches followed in the relevant literature for the Travelling Thief Problem and presenting some of the most significant methods that have been utilized by other researchers who have undertaken research into the same problem as this current work.

Chapter 4 dwells in detail on the exact methodology the present thesis has utilized in a top-down approach. Initially the solution representation that is utilised is introduced, and the way each of the the solutions to the subproblems is represented in the thesis is explained. After the representation of the solution for both the Travelling Salesman Problem and the Knapsack problem has been presented, the Objective Function (OF) of the unified Travelling Thief problem, as has been introduced in various versions from the original researchers and adapted for use in this current thesis, is elaborated in great detail. Every constituent part of the relevant components in which the TTP objective function is comprised of is meticulously explained allowing the reader to gain a deep understanding in the way the objective function is constructed upon. An illustrated example is provided, including the detailed actions for some indicative small problem with purposefully chosen convenient numbers. The entire process is elaborated with these numbers, presenting a methodical approach to solving a small Travelling Thief Problem by hand to allow the reader to deeply engage with the way these components interact in the objective function and make clear how the interdependency of the two component problems is achieved through the intertwining of their respective elements that comprise the related OF parts.

As VNS utilizes neighborhoods of solutions as integral components of the method, the selections of three specific neighborhoods that have been accepted and incorporated into the current approach are demonstrated. The exact modification that each of the neigh-

borhoods is introducing to the solution is presented and indicative practical examples of deliberately small 4x4 and 5x5 solutions are demonstrated to enhance the quality of the explanation.

The chapter concludes by elaborating into the construction function heuristic approaches that have been utilised. Three main construction heuristics have been chosen for the construction of an initial solution that will serve as further input into being the main starting point for the RVNS metaheuristic which will improve upon these initial solutions. While a random heuristic has also been used to construct initial solutions during the testing phases of the algorithm, it has ultimately been dropped from the thesis as the initial solutions it provided were exceptionally bad and have almost without fail always led to rapid improvement but with terrible final outcomes in the set number or iterations of the algorithm. The other two construction function heuristics that have been used, namely the Greedy and GRASP approaches, have yielded significantly better outcomes and are the main construction functions utilized in obtaining the results. However the results of the random initial solution approach are also presented as an aid to the reader and to demonstrate how a sufficiently bad initial solution can hamper any heuristic, regardless of its own efficiency, to a significant degree.

Chapter 5 presents the results of the experiments. Six characteristic datasets from the Adelaide Travelling Thief competition are utilised. These datasets have been purposely selected to be representative of a very wide variety of problems in order to demonstrate the efficiency of RVNS in various conditions and with a multitude of factors involved. This is also significantly important because the variations in the number of components also greatly affect the speed of the construction heuristics which can become a significant and important issue to consider when dealing with larger datasets. For an objective and extensive approach to properly demonstrate the efficiency of the method, the datasets were chosen to include either few or many cities, with a fixed number of items per city unique in each dataset as well as taking into account the degree of correlation between the value and weight of the items. Performance charts illustrating the performance of all methods, including each separate construction heuristic in the total performance for each of the datasets are presented with some commentary on the specifics of every graph. Because RVNS has an inherit component of randomness and especially in the case where the GRASP heuristic is utilised, which also incorporates stochastic elements, in order to enhance the reproducability of the results and avoid the pitfall of accidentally accepting solution that happened to be exceptionally favorable because of randomness alone each of

the experiments was run five times to better normalise the outcomes of these stochastic processes. A unified table presenting the results of these five combined runs for each of the methods is located after the graphs to better illustrate how each of the datasets compares to the others using these combinations.

Chapter 6 presents a statistical analysis of the results in a smaller category of datasets detailing the observed improvement per dataset and breaking down the individual datasets on the basis of the initial construction function used to also demonstrate the effect of proper selection of a construction heuristic in enhancing the solutions. The effectiveness of the method in each dataset is debated and presented in order to compare the suitability of RVNS and each construction heuristic to the size and degree of correlation in each dataset.

Chapter 7 presents the concluding remarks of the thesis, establishes the comparative quality of the method compared to other results for the same datasets utilised in the relevant research literature and advocates for the importance of further research into combinatorial optimization problems and the necessity of implementing dynamic and efficient heuristics that will adapt to the demanding nature of these uniquely interesting and practical problems. Admittances for the limitations of the current study are presented to the readers and a line of future work related to both the applications of RVNS in combinatorial optimization as well as the construction function selection and building process is proposed.

CHAPTER 2

Theoretical Background

In this chapter, we briefly discuss how Combinatorial Optimization problems are related to Decision Problems and introduce the issue of computational complexity. Subsequently we proceed to examine how the Knapsack problem is created from the Cutting Stock Problem and the Bin Packing Problem and advance to introduce the Travelling Salesman Problem and some of its variants.

2.1 Combinatorial Optimization Problems

The Entscheidungsproblem, meaning "Decision Problem" in German, was a question posed by David Hilbert, one of the most influential modern mathematicians, in 1928 in The International Congress of Mathematicians. The essence of Hilbert's question was whether Mathematics was "decidable", the question posed was:
"Is there an algorithm which when fed any statement in the formal language of first-order arithmetic, determines in a finite number of steps whether or not the statement is provable from Peano's axioms for arithmetic, using the usual rules of first-order logic?"

In 1936, both Church and Turing were able to prove that the Entscheidungsproblem posed by Hilbert is unsolvable. There are therefore two basic types of problems pertaining to their fundamental nature, problems that can be characterized as decision problems and undecidable problems. In undecidable problems, it is impossible to construct an algorithm that always leads to a correct yes-or-no answer in a finite time.

In fundamental computational complexity theory, decidable problems are further subdivided in multiple complexity classes based on their related resource-based complexity. Two fundamental complexity classes are P, the class of problems that can be solved by a deterministic machine in polynomial time, and NP, the class of problems that can be

solved by a non-deterministic machine in polynomial time (Hoos and Stützle 2004) and (Huang, Lai, and Cheng 2009). P is contained in NP in the sense that a non-deterministic machine can be used as a deterministic machine and the hardest problems in NP are called NP-complete. A search problem X is NP-Hard if for some NP-complete problem Y there is a polynomial-time Turing reduction from Y to X. An obvious consequence of the definition above is that that an NP-hard problem cannot be solvable in polynomial time unless P = NP (Leeuwen, Meyer, Nival, et al. 1990). Please note that while below we will be examining problems that are indeed considered NP-hard, for the purposes of this research thesis we will be ignoring the P versus NP problem, since the practical computational complexity of the problems examined is enormous.

While the class of NP is limited to decision problems and applies directly to them, a decision problem can be turned into an optimization problem by introducing a threshold value. From a complexity standpoint, the decision problem is "no harder" corresponding to the equivalent optimization problem.

Single-objective optimization problems are the simplest form of optimization problems encountered: Depending on the constraints imposed, the researcher aims to find the best solution in terms of minimum or maximum value to the modeled objective function describing the solution to the problem. Some problems are more linear while others are non-linear and their solution requires a more advanced approach and more detailed problem modeling.

While the study of single-objective optimization problems is a precious academic resource, **multi-objective** optimization is more a appropriate tool for approaching real-world problems. Indeed problems encountered in a practical scenario rarely lend themselves to optimizing only a single objective. Furthermore the very type of some problems is contradictory, they are not always single-scope problems (minimization or maximization), but very often are a mixture blending minimizing one part while maximizing another, in an intricate balance that requires a well-organized approach.

2.1.1 The Cutting Stock Problem (CSP) and the Bin Packing Problem (BPP)

The Cutting Stock Problem is a simple problem that gives rise to some of the more interesting problems we will be dealing with in this thesis. It is however valuable to give an insight into how many of the practical problems encountered in real-world conditions

are linked and interconnected among them. Given a starting stock material, the classical CSP asks for finding a way of cutting standard sized pieces from that starter material minimizing the material wasted in the process. Whether it is cutting boards from a log, or pieces of clothing from fabrics, the aim is always to minimize the material wasted, which simultaneously leads to the maximization of the material utilized.
In mathematical terms, the formulation of the problem would be as follows:

Starting with a list of $\boldsymbol{m}$ orders, each requiring $\boldsymbol{q_j}$ pieces, where $j \in$ (1,2,...,m). If $\boldsymbol{n}$ is the number of all possible combinations of cuts. Mapping $x \in \mathbb{N}^\star$ to each combination where $\boldsymbol{i}$ indicates the number of times each combination will be utilized and $i \in (1, 2, ..., n)$ then:

$$min \sum_{i=1}^{n} c_i x_i$$

$$s.t. \sum_{i=1}^{n} a_{ij} x_i \geq q_j, \qquad \forall j \in (1, 2, ..., m)$$

$$x_i \in \mathbb{N}^\star$$

where $\boldsymbol{a_{i,j}}$ is the number of times order $\boldsymbol{j}$ appears in combination $\boldsymbol{i}$ and $\boldsymbol{c_i}$ is the cost (frequently considered as waste) of combination $\boldsymbol{i}$

By replacing the constrain for the quantity from an inequality to an equality and setting $\boldsymbol{c_i} = 1$ to minimize the use of the source material the Cutting Stock Problem is converted into a different problem, a special case called the Bin Packing Problem.

The Bin Packing Problem (BPP) is one of the most well-known optimization problems: in the generalized form we are given a number of items and bins that can hold up to a specific weight or volume. the goal is to assign the items in the bins so as no bin will contain any items with weight/volume more than c and the total number of bins used is minimized.
A formalization of the Bin Packing problem can be found in [here goes the book in the comment]

Given n items and n knapsacks, with

w_j=*weight* of item j

c = *capacity* of each bin

assign each item to one bin so that the total weight of the items in each bin does not exceed c and the number of bins used is a minimum.

$$\text{minimize } z = \sum_{i=1}^{n} y_i$$

$$s.t. \sum_{j=1}^{n} w_j x_{ij} \leq c y_i, i \in N = \{1, 2, ..., n\}$$

$$\sum_{i=1}^{n} x_{ij} = 1, j \in N$$

and y_i, x_{ij} are binary variables where:

$$y_i = \begin{cases} 1 & \text{, if bin i is used} \\ 0 & \text{, if bin i is not used} \end{cases}, \qquad x_{ij} = \begin{cases} 1 & \text{, if item j is assigned to bin i} \\ 0 & \text{, otherwise} \end{cases}$$

In the special case where the number of bins are limited to one, and given a value metric to each item, the BPP is referred as the Knapsack Problem, which will be presented next.

2.1.2 The Knapsack Problem

The knapsack problem is one of the most studied problems of combinatorial optimization, mostly because of the enormous practical application in the field of logistics and transports. Essentially, a knapsack of finite capacity (usually in terms of weight) is given, along with items of different value and weight. The aim is to pick the combination of items maximizing the total value under the constraint that it still fits in the bag.

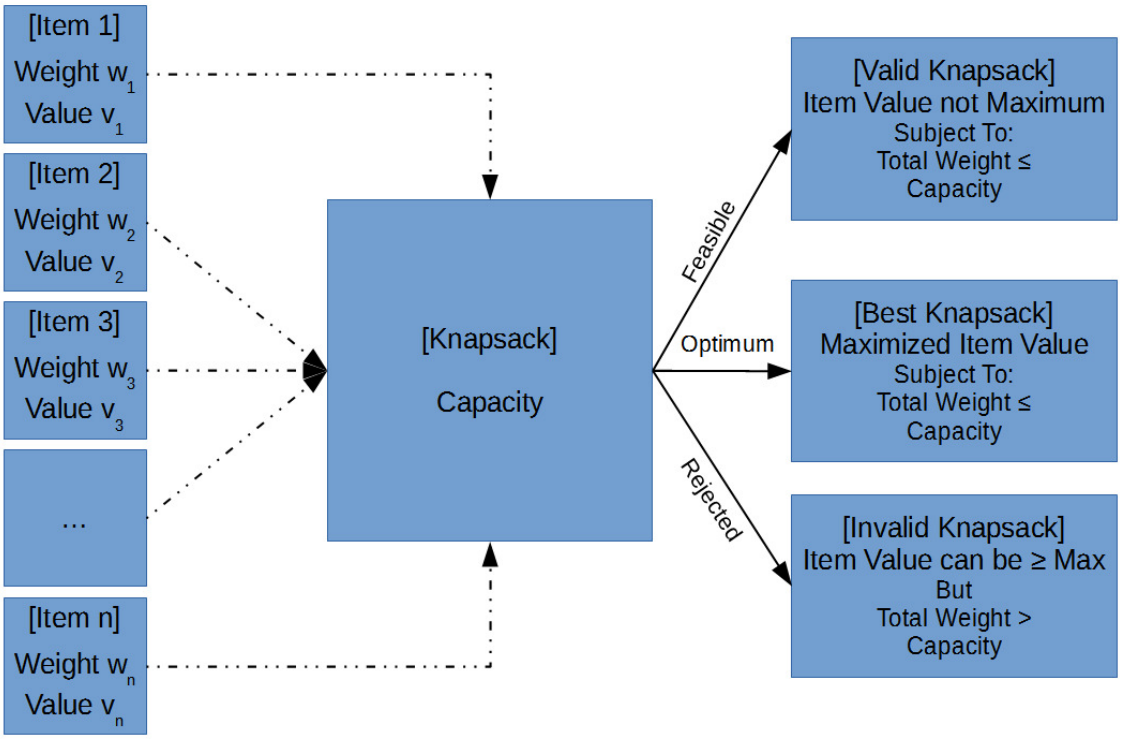

Figure **2.1.1**: A graph of the Knapsack problem showing the process and possible outcomes

While the decision problem for KS is NP-complete, the optimization problem is NP-hard. A commonly occurring variation of the common KS problem includes defining the capacity by volumetric limitations (alone or in conjunction with weight), this can be further generalized into the multi-dimensional knapsack problem where a knapsack has dimensional limitations in each dimension and the items themselves are described by vectors of said dimensions. In the latter case, as each item can be rotated on a d-dimensional axis, loading is significantly complicated and a polynomial-time approximation scheme does not exist unless P=NP.

We can easily model the integer linear programming approach to the binary form of

the Knapsack problem for single items:

Given n cities and n items

$$\text{maximize} \quad \sum_{i=1}^{n} p_i x_i$$

subject to:

$i, n \in \mathbb{N}$

While an integer linear programming approach to the multidimensional Knapsack problem can be formulated as follows:

$$\text{maximize} \quad \sum_{j=1}^{n} p_j x_j$$

subject to:

$$\sum_{j=1}^{n} w_{ij} x_j \leq c_i, \qquad i = 1, ..., m;$$

$$x_j \in \{0, 1\}, \qquad j = 1, ..., n;$$

$$p_j, w_{i,j}, c_i \in \mathbb{Z}^+$$

$$w_{ij} \leq c_i, \qquad j = 1, ..., n;$$

$$\sum_{j=1}^{n} w_{ij} \geq c_i \qquad i = 1, ..., m;$$

where:

p_j : profit of project j

w_{ij} : consumption of project j from resource i

c_i : capacity of resource i

$$x_j : \begin{cases} 1 & \text{, if project j is selected} \\ 0 & \text{, otherwise} \end{cases}$$

2.1.3 The Travelling Salesman Problem

One of the most studied problems in combinatorial optimization encountered within the field of Operations Research is the Travelling Salesman Problem (TSP). In its quintessential form, a traveling salesman is given a starting city, i.e. starting node, a set of other cities to visit and the intercity distances, and then has to decide which is the shortest route that crosses each city exactly once and ends back to starting city. The aim of the problem is minimizing the distance travelled, expressed in terms of cost.

Typically, distances are stored into a matrix as shown below:

	A	B	C
A	0	c_{ab}	c_{ac}
B	c_{ba}	0	c_{bc}
C	c_{ca}	c_{cb}	0

Table **2.1**: The generic distance matrix for 3 cities

	A	B	C	D
A	0	c_{ab}	c_{ac}	c_{ad}
B	c_{ba}	0	c_{bc}	c_{bd}
C	c_{ca}	c_{cb}	0	c_{cd}
D	c_{da}	c_{db}	c_{dc}	0

Table **2.2**: The generic distance matrix for 4 cities

It is easily observable that in case the TSP is symmetric, the matrix is also symmetric, i.e. $A = A^T$. In more detail, the mathematical formulation of the problem is as follows:

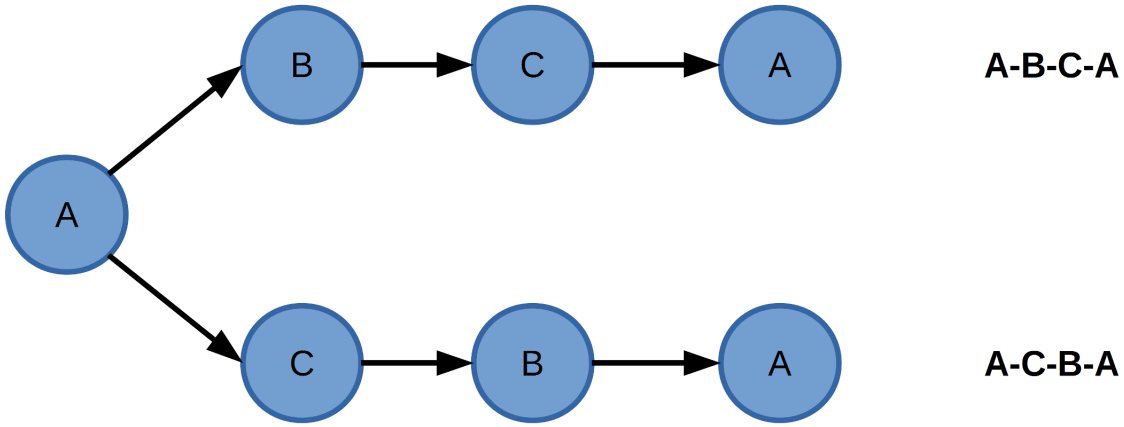

Figure **2.1.2**: A tree graph showing all possible paths in a 3 node TSP problem

Compared to the figure above, the tree in figure **2.1.3** below only has one more node, yet the increase in complexity is becoming obvious:

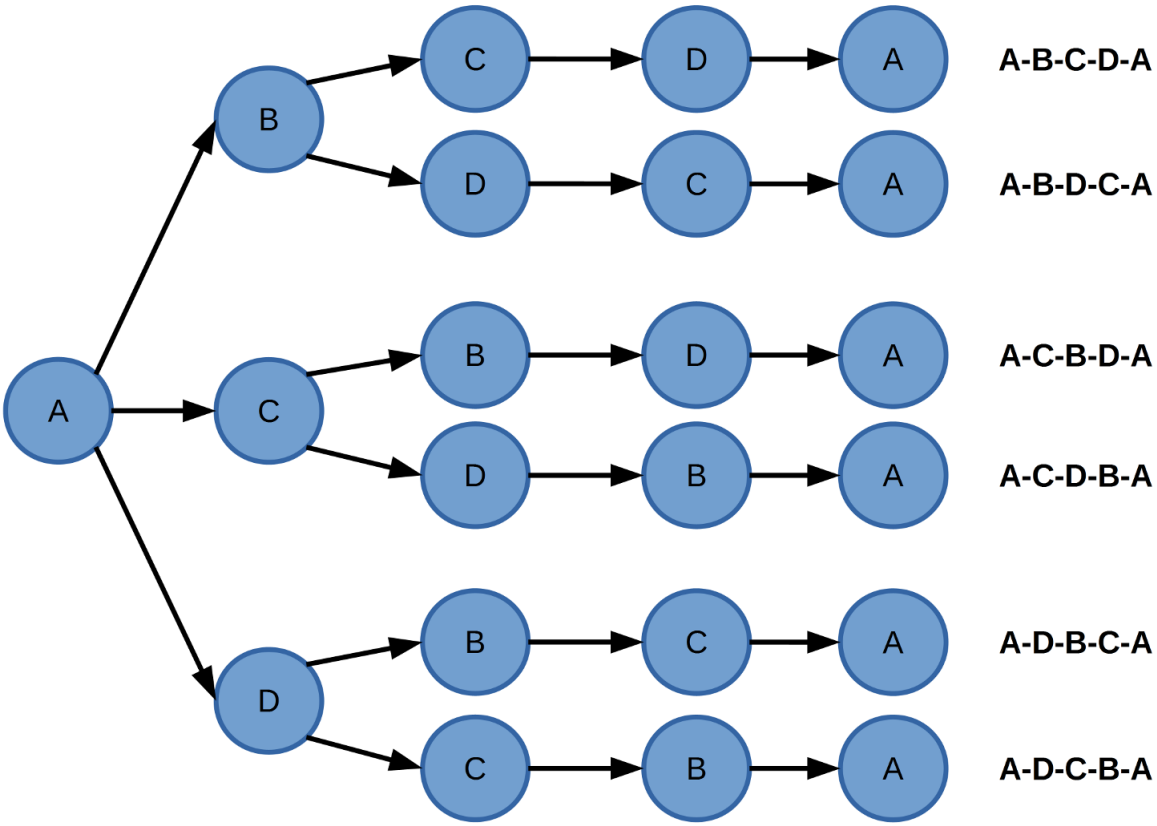

Figure **2.1.3**: A tree graph showing all possible paths in a 4 node TSP problem

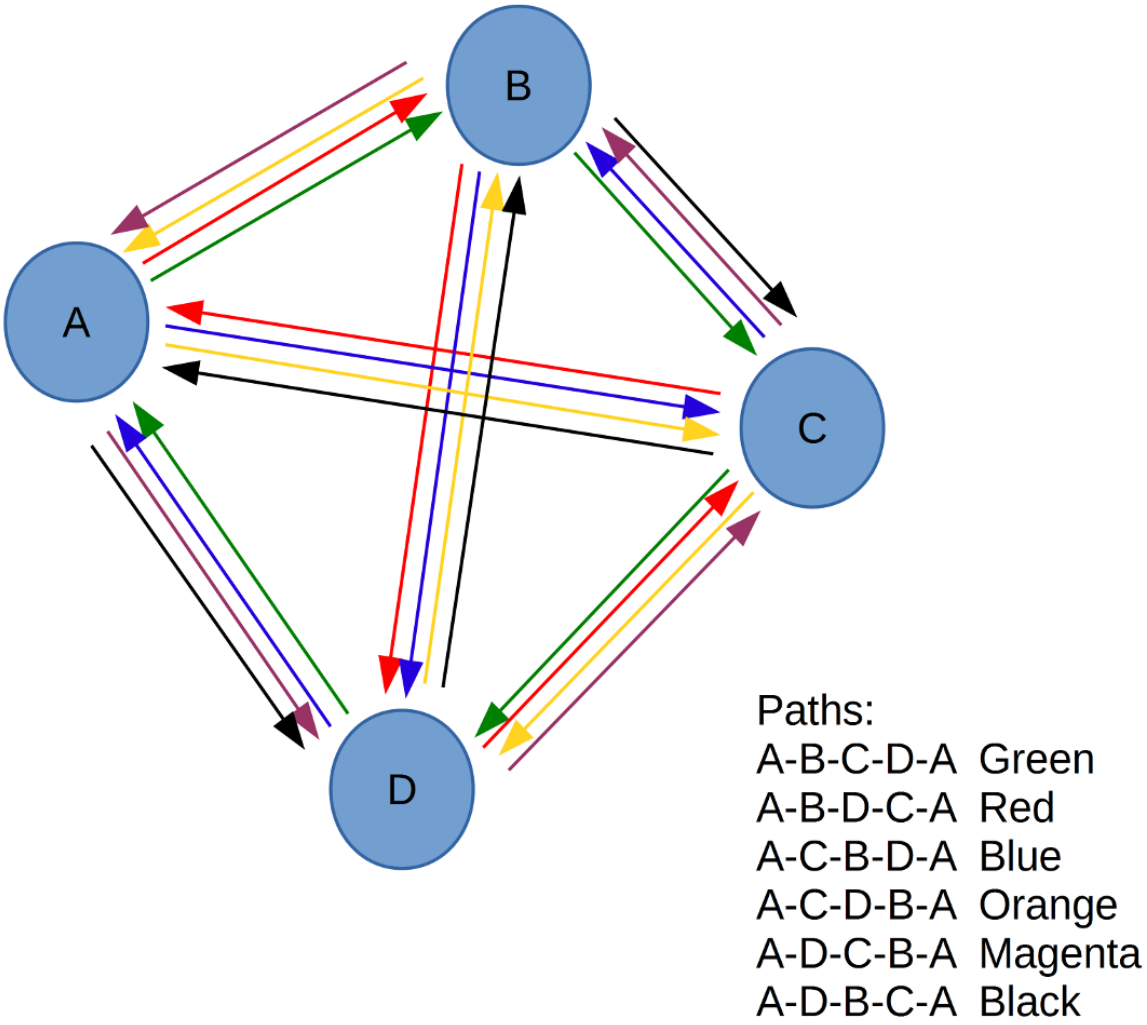

Figure **2.1.4**: An illustration depicting all possible routes between four nodes, always beginning and ending at node A

Integer Linear Programming Formulations:

These are two popular formulations of the TSP in an Integer Linear Programming Form encountered in relevant literature:

- *The Miller-Tucker-Zemlin formulation*

$$x_{ij} = \begin{cases} 1 & \text{, the path goes from city i to city j} \\ 0 & \text{, otherwise} \end{cases}$$

For $i=1,2,\ldots,n$, let u_i be a variable, and finally take c_{ij} to be the distance from city i to city j. Then the TSP can be written as the following integer linear programming

problem:

$$
\begin{aligned}
&min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij} : \\
&0 \leq x_{ij} \leq 1 && i, j = 1, ..., n; \\
&u_i \in \mathbb{Z} && i = 1, ..., n; \\
&\sum_{i=1, i \neq j}^{n} x_{ij} = 1 && j = 1, ..., n; \\
&\sum_{j=1, j \neq i}^{n} x_{ij} = 1 && i = 1, ..., n; \\
&u_i - u_j + n x_{ij} \leq n - 1 && 2 \leq i \neq j \leq n; \\
&0 \leq u_i \leq n - 1 && 2 \leq i \leq n.
\end{aligned}
$$

- *The Dantzig-Fulkerson-Johnson formulation*:

  Label the cities with the numbers 1,...,n and define:

$$
x_{ij} = \begin{cases} 1 & \text{, the path goes from city i to city j} \\ 0 & \text{, otherwise} \end{cases}
$$

  Take c_{ij} to be the the distance from city i to city j. Then the TSP can be written as the following linear integer programming problem:

$$min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij} :$$

$$0 \leq x_{ij} \leq 1 \qquad i, j = 1, ..., n;$$

$$\sum_{i=1, i \neq j}^{n} x_{ij} = 1 \qquad j = 1, ..., n;$$

$$\sum_{j=1, j \neq i}^{n} x_{ij} = 1 \qquad i = 1, ..., n;$$

$$\sum_{i \in Q} \sum_{j \in Q} x_{ij} \leq |Q| - 1 \qquad \forall Q \subseteq \{2, ..., n\}$$

2.2 Meta-heuristics in Combinatorial Optimization

In this subsection we delve into some of the more popular metaheuristics utilized in combinatorial optimization approaches, present their main features and examine some of the drawbacks for each method.

2.2.1 Population Based

Genetic Algorithms A Genetic Algorithm (GA) is a meta-heuristic in the class of evolutionary algorithms that was introduced in 1960 by John Holland. The concept of a GA is to represent the solution of an optimization problem as genes and to approximate the solution by continuous evolution through inheritance and mutation. By creating sets of "parents" and deriving "children" from the combinations of their genome. Enabling some degree of gene mutation during the generation of children allows leaps in various directions to explore the possibility of finding a better solution elsewhere in the space of all feasible solutions and allowing the mutated genes of a successful subject to spread.

To gain a more complete insight in this context, it is important to reference some of the some distinctive disadvantages GAs possess compared to alternative optimization algorithms:

- Complexity scaling is difficult with Genetic Algorithms. In particular, where the number of elements which are exposed to mutation is sufficiently extensive, there

is often an exponential increase in search space size. This makes it extremely difficult to use the technique on problems with multicomponent interdependent variables. To facilitate evolutionary search approach in such problems, they must be reduced into as simple a representation as possible. A related limitation is the issue of protecting genome parts that have evolved to represent good solutions from detrimental mutations, particularly when there are imposed fitness limitation that require them to combine with other parts.

- Rating a solution as being "better" is implemented in comparison to other solutions resulting in the stop criterion not being straightforward in every problem.
- GAs have difficulty sacrificing short-term fitness to balance the gains in longer-term fitness. The critical factor in determining the likelihood of this scenario is the shape of the fitness landscape.
- Genetic Algorithms face difficulties operating on dynamic data sets. Early premature convergence may derail the algorithm towards operating on solutions which may no longer be valid for data acquired subsequently. This is often countered by increasing genetic diversity either through a method called triggered hypermutation or by periodically introducing elements termed random immigrants, which are new, randomly generated elements, into the gene pool.
- In practical terms, GAs are inefficient in solving decision problems, problems where the only fitness measure is a binary true/false measure. This is because there is no way to converge on the solution though hill climbing.
- The suitability of Genetic Algorithms, as for all optimization algorithms, is not universal. While genetic algorithms are generally efficient, advanced understanding or design of the components of a problem may facilitate the use of other optimization approaches, or even hybrid approaches, that converge faster and more efficiently.

Particle Swarm Optimization Particle Swarm Optimization (PSO) is an advanced population based meta-heuristic for optimizing a solution through iterative improvement of candidate solutions (particles), which are individually characterized by their position and velocity. The principle behind this approach lies in the utilization of the swarm movement of particles to converge towards local minima in the search space, interactively influencing the swarm around them while being influenced by it themselves.

Some disadvantages of PSO have been noted by (Abdmouleh, Gastli, Ben-Brahim, Haouari, and Al-Emadi 2017), which focus their criticism on three key points:

- Inherent difficulty in defining the initial design parameters
- Significant difficulty in dealing with problems of Scattering
- Potential for prematurely convergence and entrapment in local minima, especially with complex problem

Their third point, premature convergence, is of course a universal challenge faced by the entirety of heuristics and metaheuristics as the lack of complete information on the terrain of the search space in conjunction with imperfect convergence criteria inherent to all stochastic heuristic processes possesses the inherent risk of trapping algorithms in local minima.

Ant Colony Optimization Ant Colony Optimization (ACO) was proposed by M. Dorigo in his PhD thesis during the '90s and published in (Dorigo and Caro 1999). It is a nature-inspired, stochastic, swarm-intelligent metaheuristic and the inspiration for this method came from the natural behavior of ants when they are looking for food.

Essentially the method is inspired by the approach ants take to finding short paths between sources of food and their colony. When ant scouts detect a potential source of food, they return to their nest while leaving a pheromone trace on the way back. The other ants are attracted by the smell of the pheromone and choose to follow a path set by one of the scouts based on the intensity of the smell. The ants taking the longer path will return later than the ants taking a shorter path and hence the levels of evaporation of the pheromone trail will be less for shorter paths and more for longer paths. Hence shorter paths will gain a stronger pheromone trail and lead more ants to prefer them over time, creating a positive feedback loop that creates a preference for the shortest path.

The mathematical modelling of the ACO algorithm for the TSP that is presented below allows a greater understanding of the ACO principle. It was proposed by (Stützle, Dorigo, et al. 1999) in their paper *'ACO Algorithms for the Traveling Salesman Problem'* and focuses on the Ant System (AS). For the tour construction, each of the starting ants is initially placed at random in a chosen city. For each next step, the ant applies a probabilistic action choice rule to move to a city using the following formula:

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^a \cdot [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}(t)]^a \cdot [\eta_{il}^\beta]} \qquad \text{if } j \in \mathcal{N}_i^k$$

The parameter α corresponds to the selection of closer cities by the ants, acting like a classic greedy heuristic, while parameter β affects the pheromone amplification. An improper balance of these parameters will result in stagnation by rapid convergence to sub-optimal solutions.

After ants have constructed their tours, the pheromone trail strength is lowered in all arcs and each ant is adding pheromone to the arcs it has visited based on the formula below:

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^k(t) \qquad \rho \in (0,1]$$

Where

$$\Delta\tau_{ij}^k(t) = \begin{cases} \frac{1}{L^k(t)} & \text{, if arc (i,j) is used by ant k} \\ 0 & \text{, otherwise} \end{cases}$$

The parameter ρ deals with the evaporation of the pheromone trails. The pheromone strength in arcs which were not chosen decreases exponentially. The $\Delta\tau_{ij}^k(t)$ part is the amount of pheromone ant k puts on the arcs it has visited while $L^k(t)$ refers to the length of the kth ant's tour. From the equation above defining the amount of pheromone, it is easy to observe that the better tours receive more pheromone and hence the probability of them being chosen from other ants in later iterations is greater.

However, according to the authors the Ant System provides rather poor solution quality for TSP instances with over 75 cities and later research has been mostly focused on improving ACO over the initial form of AS. One such notable improvement, the Ant Colony System (ACS) was later introduced to improve the performance of Ant System, based of a previous algorithm called Ant-Q. According to the authors, the Ant System and the Ant Colony System approaches differ in three key areas. The implementation of a more aggressive action choice rule, adding the pheromone only in arcs belonging to the global-best solution and ants removing some pheromone from each arc they choose to use.

2.2.2 Single Point Search

Some of the more popular heuristic methods, frequently applied to real-world problems, are Single Point Search meta-heuristics, also known as trajectory-based meta-heuristics. The name comes from the characteristic pattern of the trajectory they follow in the search space. This is usually one of the two main categories used to subdivide meta-heuristic methods based on their approach, the other being population-based methods which were described above. Below we are introducing some of the most popular single point search methods encountered in published research with their distinctive characteristics and drawbacks, and ending the section with the methods applicable to the implementation proposed in this thesis.

Simulated Annealing is a very interesting and unique among meta-heuristics, the approach to this method is frequently being cited to have been inspired by the metonym process in the craft of metallurgy. The basic premise is heating a material to a point above its recrystallization temperature, and exposing it to a slow and controlled cooldown, keeping the temperature at a controlled rate with the aim to produce a crystalline structure with particular properties.

Simulated Annealing implements the same concept by utilising a factor introduced as temperature, which is essentially a plasticity factor. This term controls the varying probabilities of a) accepting a new worse solution and b) accepting a new better solution while navigating the search space. Temperature is gradually decreased, altering these two probabilities, with case (a) gradually decreasing while case (b) gradually increasing.

One interesting element that sets this method apart is that it is very well suited to estimating the global optimum, making it valuable in cases where an approximation of the position of the global optimum is more important than finding a good-enough solution that might happen to be a local minimum. However there are important limiting factors to Simulated Annealing that need to be considered. Repeated annealing is slow and compared to other methods is disproportionately affected by the complexity of the objective function. Furthermore it is less suited to exploring relatively "smooth" areas where the positions of local minima are located inside dense regions of uniformly low fitness.

Algorithm 1: A pseudocode for Simulated Annealing, (Ferentinos, Arvanitis, and Sigrimis 2002)

```
Initialize (temperature T, random starting point)
while cool_ iteration <= max_iterations do
    cool_iteration=cool_iteration+1
    temp_iteration=0
    while temp_iteration<=nrep do
        temp_iteration=temp_iteration+1
        select a new point from the neighborhood
        compute current cost (of this new point)
        δ = current_cost - previous_cost
        if δ < 0 then
            accept neighbor
        else
            accept with a probability exp(-δ/T)
        end
    end
    T = a*T (0 < a < 1)
end
```

Tabu search is a metaheuristic method utilizing local search methods. A defining characteristic of this method is the introduction of prohibitions restricting the algorithm's ability to visit past solutions.

Essentially Tabu search utilizes a common approach in local search, that is, after establishing an initial solution, it proceeds to further examine neighboring solutions: neighboring being defined as solutions that are quite similar in their structure except for some minor differences, relating to the initial problem and the representation of the solution. Like most metaheuristic methods, it initially allows for some tolerance when it comes to accepting potentially 'worse' (less fit) solutions to facilitate escaping from local minimum traps and broaden the scope of the search in the field of the solutions space.

A problem-specific memory structure, often referred simply as the 'tabu list', is a set of previously visited solutions or neighborhoods of solutions and defined rules that determine the behavior of the algorithm. An item that in some previous iterations has been introduced into the list is considered to be 'tabu' (prohibited) and the algorithm does not consider it a viable option during the evaluation. In general the status of the a solution being in the tabu list is dynamic, allowing them to enter and exit the structure often and according to the queuing approach that has been selected for the structure, the

implementation of which can vary.

Pseudo code for TS algorithm
Initialization of algorithm
Generate an initial solution (S_0) randomly or using some heuristics. Define neighbourhood generation scheme, tabu list size (TL), aspiration criterion and termination criterion (ITR_{Max})
Set Best solution (S_{Best}) = Current solution (S) = Initial solution (S_0)
Tabu list (TL) = {}
Iterations (ITR) = 0
If $ITR \neq ITR_{Max}$,
$S \rightarrow S_N$ (generate neighbourhood solutions)
For $S \in S_N$
If $S \notin TL$
If (fitness (S) > fitness (S_{Best}))
$S_{Best} = S$
End
End
End
End
Return S_{Best}

Figure **2.2.5**: An indicative pseudocode for Tabu Search

Another single-point search method utilizing local search is the Variable Neighborhood Search metaheuristic. VNS and one of it's variants, Reduced VNS (RVNS), constitutes the foundations of our approach to the TTP and as such it is further examined in detail below.

2.3 Variable neighborhood search

Variable Neighborhood Search, abbreviated as VNS, is an advanced metaheuristic for solving Combinatorial Optimization and Global Optimization problems (Cafieri, Hansen, and Mladenović 2014). It was proposed by Mladenović and Hansen in 1997 and has since been incorporated into a large volume of research by various authors in the fields of heuristics, optimization and operational research.

According to (Hansen, Mladenović, and Pérez 2010), VNS exploits systematically the following observations:

Fact 1 A local minimum with respect to one neighborhood structure is not necessary so for another;

Fact 2 A global minimum is a local minimum with respect to all possible neighborhood structures.

Fact 3 For many problems local minima with respect to one or several neighborhoods are relatively close to each other.

Surmising from the latter point, which according to the same paper is an empirical observation, that some information about the global optimum can be often be provided by a local optimum.

While VNS belongs to the class of local search heuristics like Simulated Annealing and Tabu search which were referenced above, unlike them it is not a trajectory based method[1] although it has mistakenly been claimed to be in some publications. Rather, the core concept of VNS is to pick an initial solution, discover a direction of descent from that solution (assuming the problem has been converted into a proper minimized form) within a neighborhood of that solution, and descend towards the minimum value of the evaluated function within that neighborhood. A perturbation phase follows the descent phase to enable the algorithm to escape from the corresponding 'valley', if we assume a graphical representation of a local minimum to be a depression point. It is interesting to note that no prior knowledge of the landscape is assumed or exploited in VNS. Furthermore it has to be noted that local search in problems with very large instances is very computationally intensive and might not be an intuitively efficient approach.

A summary of the way VNS operates was written in (Pérez, Mladenović, Batista, and del Amo 2006) in the journal of *Metaheuristic Procedures for Training Neural Networks*, and bears exceptional importance because of the contribution of Mladenović, the inventor of VNS:

"*VNS proceeds by a descent method to a local minimum exploring then, systematically or at random, increasingly distant neighbourhoods of this solution. Each time, one or several points within the current neighbourhood are used as initial solutions for a local descent. The method jumps from the current solution to a new one if and only if a better solution has been found.*" (Pérez, Mladenović, Batista, and del Amo 2006).

Normally, at each step the neighborhood of a solution is explored completely. However, there are factors that can make this option computationally expensive. Considerations like the complexity of the problem, whether the form of the problem is discrete or continuous, run-time parameters or even hardware limitations, all play an important part in determining whether complete exploration is needed or can be substituted for a different approach. An alternative to exploring the entire neighborhood is to use a different heuristic method to select a criterion for the descent. One such approach is the use of the first descent heuristic, suggested for that very reason from Hansen and Mladenović (Cordeau, Gendreau, Hertz, Laporte, Cordeau, and Sormany 2004) which enumerates vectors x' in the neighborhood N(x) of the solution and immediately selects the first to descend.

The steps which describe the operating principles of VNS as pertaining to a minimization problem are presented in the following pseudocode:

Algorithm 2: VNS pseudocode for a minimization problem

```
initialize solution x
while stopping criteria are not met do
    k = 1
    while k ≤ k_max do
        generate x' a random solution from neighborhood N_k(x)
        x'' = localSearch(x')
        if evaluate(x'') < evaluate(x) then
            x = x''
            k = 1
        else
            k = k + 1
        end
    end
end
return x;
```

According to (Cordeau, Gendreau, Hertz, Laporte, Cordeau, and Sormany 2004) the main ingredients of the VNS metaheuristic are:

- Definition of a neighbourhood of the current solution;

- Neighbourhood changes;
- Local search;
- Shaking, i.e., a procedure to perturb the current solution;
- Update of the current solution

2.3.1 Variants of the Variable Neighborhood Search

VNS is a very flexible metaheuristic that since its introduction has be extended in numerous ways to complement and reinforce it. Some of the extensions frequently encountered in relative literature are presented here, finishing with the actual method that has been used in this thesis.

Variable Neighborhood Descent (VND) Variable Neighborhood Descent (VND) is a variant of VNS. The concept behind this approach is based on the observation that diverse neighborhood structures have inherently different local minima and attempts to resolve this though switching among the neighborhoods. Thus the process undertaken in this variant is to explore a neighborhood until a local minimum is encountered and then switch to a different neighborhood that may allow the algorithm to proceed farther.

According to (Duarte, Sánchez-Oro, Mladenović, and Todosijević 2018), there are three possible ways to design a VND-based local search routine:

(i) Sequential VND, where neighborhoods are placed in the list with a given order and always explored in that order.

(ii) Nested or composite VND, where neighborhood operators are nested, i.e., N1(N2(N3(...(x))), which can be considered as neighborhood one of neighborhood two of neighborhood three, etc. of x.

(iii) Mixed Nested VND, a hybrid approach of the two methods above where the two previous strategies are combined.

Algorithm 3: A pseudocode for VND by (Lusa and Potts 2008)

```
x is the initial solution for VND
while no final condition do
    u = 1
    while u ≤ u_max do
        x' is the best solution in N_u(x)
        if f(x') < f(x) then
            x = x'
            u = 1
        else
            u = u + 1
        end
    end
end
Return S
```

VND is often utilized as a local search function within other metaheuristics.

Variable Neighbourhood Decomposition Search (VNDS) According to (Hansen, Mladenović, and Perez-Britos 2001) Variable Neighbourhood Decomposition Search is a two-level VNS where the main VNS method is enhanced by decomposition. Essentially

the main VNS method is run within a successive approximations decomposition method.

Algorithm 4: VNDS pseudocode (Hansen, Mladenović, and Perez-Britos 2001)

Initialization: Select the set of neighborhood structures $\mathcal{N}_k$, $k = 1, ..., k_{max}$ that will be used in the search;
find an initial solution x; choose a stopping condition;
while *stopping criteria are not met* **do**
  Set $k \leftarrow 1$
  while $k \neq k_{max}$ **do**
    (a) Shaking: generate a point x' at random from k^{th} neighborhood of x ($x' \in \mathcal{N}_k(x)$); in other words let y be a set of k solution attributes present in x' but not in x ($y = x' \setminus x$)
    (b) Local Search: Find the local optimum in the space of y either by inspection or by some heuristic; denote the best solution found with y' and with x'' the corresponding solution in the whole space S ($x'' = (x' \setminus y) \cup y'$);
    (c) Move or not: If the solution thus obtained is better than the incumbent, move where $x \leftarrow x$ and continue the search with $N_1(k \leftarrow 1)$; otherwise set $k \leftarrow k + 1$;
  end
end
return x;

The difference between VNS and VNDS according to the authors of the heuristic is in step (b) in the internal loop of the above pseudocode, where instead of applying a local search method that would encompass the entire solution space, a subproblem in some subspace is solved at each iteration (Hansen, Mladenović, and Perez-Britos 2001)

Skewed VNS While some problems or specific datasets present with clustered local optima, this is not always not necessarily the case. While VNS gives equal or better solutions to the multistart method and even better ones in the case of multiple of local optima, in the cases where neighborhoods are larger and far apart valleys containing near-optimal solutions, VNS degenerates into multistart (Burke and Kendall 2013)

Skewed VNS is therefore a modified VNS scheme, a method for avoiding the issues created by the existence of very large valleys near x, which would otherwise create a possible entrapment scenario, infinitely recentering the algorithm to searching near a local optimum. According to (Burke and Kendall 2013), this is achieved by allowing for recentering of the search when a solution is found that is close to the current best, albeit not necessarily a better one, with the additional admission that it is situated far from it.

Algorithm 5: Pseudocode for Skewed VNS by Hansen, P. and Mladenović, N. via (Burke and Kendall 2013)

Initialization: Select the set of neighborhood structures $\mathcal{N}_k$, $k = 1, ..., k_{max}$ that will be used in the search;
find an initial solution x and its value $f(x)$; set $x_{opt} \leftarrow x$, $f_{opt} \leftarrow f(x)$;
choose a stopping condition and a parameter value α;
while *stopping criteria are not met* **do**
    Set $k \leftarrow 1$
    while $k \neq k_{max}$ **do**
        (a) Shaking: generate a point x' at random from k^{th} neighborhood of x
        (b) Local Search: Apply some local search method with x' as initial solution; denote with x'' the so obtained local optimum
        (c) Improvement or not: If $f(x'') < f_{opt}$ set $f_{opt} \leftarrow f(x)$ and $x_{opt} \leftarrow x''$;
        (d) Move or not: If $f(x'') - \alpha\rho(x, x'') < f(x)$ set $x \leftarrow x''$ and $k \leftarrow 1$; otherwise set $k \leftarrow k + 1$.
    end
end

Another very popular variant of the VNS in academic research and the very one used in this thesis is Reduced Variable Neighborhood Search (RVNS), the major difference among it and classic VNS is the introduction of stochastic elements to circumvent the need for a complete exploration of the neighborhood, making it a valuable tool for problems with large instances where exhaustive local search can be counterproductive. RVNS is one of the main components of this thesis and is examined in more detail below.

2.3.2 Reduced Variable Neighborhood Search

Reduced Variable Neighborhood Search (RVNS) is a stochastic variant of the basic VNS heuristic. The basic premise of the RVNS approach focuses on introducing elements of randomness in the exploration of a series of neighborhoods by starting with an incumbent solution x, evaluating a stochastically chosen solution x' in the first neighborhood and when focusing the local search around this new solution if it is value is evaluated to be a better fit than the current i.e., if $f(x') < f(x)$ for minimization problems. This is based on the premise that in many optimization problems, there is some empirically observable congregation of local optima, which tend to be situated in close proximity to one another and situated in one or several small parts of the search space. The neighborhoods considered by RVNS are often nested, focusing the search closer to neighborhoods N(x) of x, rather than others further away.

Essentially RVNS introduces an element of stochasticity in the basic VNS scheme, enabling the exploration of nested neighborhoods and utilizing the realization that in many problems local optima tend to congregate. An obvious observation is that unlike VNS, RVNS does not utilize local search and instead replaces it with local stochastic jumps. This is especially valuable in very large instances where local search can be very computationally costly.

Choosing the direction of movement differs depending on the type of the problem. In 0-1 variable type problems, like the Knapsack problem which is examined in this thesis, simple complementation of some variables can be enough. In permutation problems, like the TSP, changing the direction of movement may be done with simple swaps but depending on limitations (e.g., in the directed TSP problem) some solutions might be infeasible and thus outside the search space. In continuous Euclidean problems, selecting a random spot in a sphere surrounding x and using that to create a vector that enables that construction of random angles is another way to incorporate stochastic elements in the direction of movement.

Having mentioned above that RVNS places a strong emphasis on the exploration of neighborhoods close to x, exploring its vicinity first, one may wonder on effective ways to deal with cases such as when a very large 'valley' surrounds the local optimum, or when

The pseudo-code of the flow of RVNS for a minimization problem is presented below:

Algorithm 6: RVNS pseudocode for a minimization problem by (Resende and Ribeiro 2010)

```
initialize solution x
while stopping criteria are not met do
    k = 1
    while k ≤ k_max do
        generate x' a random solution from neighborhood N_k(x)
        if evaluate(x') < evaluate(x) then
            x = x'
            k = 1
        else
            k = k + 1
        end
    end
end
return x;
```

2.4 Construction heuristics

On a fundamental level we can distinguish three basic types of functions in heuristic methods:

- Construction Heuristics
- Improvement Heuristics
- Hybrid Heuristics

The object of pure construction heuristics is to determine an initial solution to the problem. The extent to which this solution will be accepted as-is or be passed on for further improvement frequently determines the type of heuristic method selected as the initial construction heuristic. In a theoretical sense, even a random walk can be a construction heuristic, producing solutions very quickly with little regard as to the quality of

the output. More advanced construction heuristics require more time but as a rule tend to produce better solutions. In practice, a frequently occurring approach is to implement greedy construction heuristics, i.e. methods that form a solution by taking the best decision having only partial knowledge of the solution's performance. However the drawback to this approach is that local optimality does not necessarily yield solutions close to a global optimum and may in fact lead to a path of rapid stagnation. Furthermore making the locally optimal choice can require a great number of calculations and memory space, making the use of the greedy heuristic questionably applicable the more the size of the problem increases.

Improvement heuristics on the other hand are methods that receive a solution as an input and try to improve on that solution. The quality of the initial input and the runtime parameters of the improvement method significantly effect the speed of convergence, the quality of the output and the resource utilization of the algorithm. Some improvement heuristics are significantly better suited to solving specific types of problems.

Hybrid heuristics are algorithms that incorporate their own construction and improvement heuristic, not having to rely on receiving an input of the initial solution externally. An example of this can be identified the Ant System approach which is briefly examined in this thesis and incorporates its own construction function.

2.4.1 Greedy method - Hill climbing

Among the popular techniques for traversing a space encountered in the relevant literature are the hill-climbing methods, which traverse by moving from one point to the adjacent point having the highest elevation. In essence hill-climbing is a greedy search technique, under a given evaluation function, it selects the best successor node and commits the search to it. The resulting successor serves as the actual node, and the search continues. While this method proves to be extremely efficient for some problems, it can be trapped in state space problem graphs with dead-ends, making the selection of this technique a sub-optimal choice for problems where the graph might be directed (one such case being directed or TSP). Hill-climbing methods typically terminate when there is no adjacent point having a higher elevation than the current point.

A more popular, yet more conservative form, which is also a more stable version

Algorithm 7: The pseudocode for a greedy algorithm for a minimization problem by (Resende and Ribeiro 2010)

$S \leftarrow 0$
Initialize the candidate set: $C \leftarrow E$
while $C \neq \emptyset$ **do**
    Select an element $s \in C$ with the smallest incremental cost $c(s)$
    Incorporate s into the current solution $S \leftarrow S \cup \{s\}$
    Update the candidate set C
    Reevaluate the incremental cost $c(e)$ for all $e \in C$
end
Return S

of hill-climbing search, is enforced hill-climbing. It picks a successor node, only if it has a strictly better evaluation than the current node. Since this node might not be in the immediate neighborhood of the current node, enforced hill-climbing searches for that node in a breadth-first manner. Besides being incomplete in directed graphs the algorithm has other drawbacks. There is evidence that when the heuristic estimation is not very accurate, enforced hill-climbing easily suffers from stagnation or is led astray.

The traditional greedy approach for the nearest neighbor heuristic is essentially a modified enforced hill-climbing technique where the next node with the highest elevation is the one in which the distance to the nearest neighbor is a minimum. In cases where similar distances are found, causing multiple 'peaks' to have the same value, some selection technique needs to be implemented to differentiate and select among the multiple candidates.

2.4.2 Semi greedy - GRASP

Stochastic greedy algorithms (SGA) differ from the classic greedy approach in that instead of making the locally optimal choice, they defer to a stochastic method to make that selection, utilizing a probability distribution to select the next one. When implemented as a complete heuristic, and not as a construction function alone, SGA algorithms generate a multitude of solutions, the best of which is returned as the result. The introduction of stochastic elements allow greater flexibility in dealing with escaping from local minima.

GRASP, which stands for *Greedy Randomized Adaptive Search Procedure*, is a complete meta-heuristic algorithm. The basic principle of the algorithm is to iterate through

stochastically greedy solutions while applying a local search procedure seeking to converge to a local optima. The construction function of the GRASP technique is essentially an SGA, combining greedy techniques with probabilistic elements.

The basic construction function of the GRASP approach, which is also utilized as one of the construction heuristics used in this thesis is exemplified below:

Algorithm 8: Greedy randomized algorithm pseudocode for minimization by (Resende and Ribeiro 2010)

$S \leftarrow 0$
Initialize the candidate set: $C \leftarrow E$
while $C \neq \emptyset$ **do**
    Build a list with the candidate elements having the smallest incremental costs
    Select an element s from the restricted candidate list at random
    Incorporate s into the solution $S \leftarrow S \cup \{s\}$
    Update the candidate set C
    Reevaluate the incremental cost $c(e)$ for all $e \in C$
end
Return S

Obviously the step where we select an element from the restricted candidate list at random can be modified in various ways to accommodate for different stochastic approaches. Without delving into greater detail, the approach used in this thesis for the creation of the applicable greedy construction function utilizes an attenuating probability approach where closer neighbors are given a greater potential for selection compared to the more distant neighbors, with a decreasing probability depending on their ranking from closest to furthest.

2.5 Multi-objective Optimization

While selecting only one parameter of a problem to examine is a useful approach for academic environments, real problems are rarely if ever defined by a single parameter or are limited to one objective. More commonly, real-world scenarios are an interweaved function of a number of variables that need to be examined as a unified block, each

variable influencing and being influenced by the others. As such many of the problems encountered in operations research are essentially multi-objective optimization problems and the actual optimised value for all variables might be very different than the optimal solutions for each of the sub-problems alone.

This can be demonstrated to be the case in the specific scenario being evaluated in this thesis, even under the assumption that the exact best solutions for the TSP and Knapsack problems are known. Indeed, even if the best TSP is selected, there is no guarantee that the knapsack found to be optimal when examined alone is actually optimal for that particular instance of the TSP. Likewise, even the optimal knapsack selection contributes little when the selection is such that it distorts the value of the TSP that would be optimal were it being examined in isolation. Indeed, the actually optimal solution to the combined problem would be the one that would yield the best value for a combination of both problems, depending on their definitions in the objective function and the interdependence of both variables.

Since problems met in real-life scenarios appertain to the MOOP category, extended research has been conducted upon this domain. Next, we briefly discuss two well-known methods commonly used for tackling MOOPs.

2.5.1 Weighted sum method

According to (Deb 2014), the weighted sum method scalarizes a set of objectives into a single objective by pre-multiplying each objective with a user-supplied weight. In essence, what this method implies can be seen in equation 1.

$$\begin{aligned} minimize \quad & F(x) = \sum_{m=1}^{M} w_m f_m(x) \\ & \text{subject to:} \\ & \ldots \end{aligned} \tag{1}$$

where each w_i acts as an *importance* factor on the objective f_i. Worth to note that, typically, weights add to one.

Despite the simplest one and most widely used, the author points out some of the said method's drawbacks, also. In more detail, setting the value of w_i's is a tricky, error-prone procedure and, in fact, imposes an extra optimization problem, that of choosing the

best weight values. Furthermore, concerning problems with a non-convex Pareto optimal front, more adversities can possibly come out. The interested reader is referred to (Deb 2014) for more information on this topic.

2.5.2 ϵ - Constrained method

The ϵ - Constrained method resulted from (Haimes 1971). The proposal in this research work was to keep only one objective in the objective function and restrict remaining objectives,i.e., f_is, by some parameter ϵ_i. Essentially, ϵ_is act as upper or lower bounds for f_is and, as a consequence, turn the ϵ - Constrained method suitable for problems with both a convex and a non-convex Pareto optimal front. However, the solution to a problem defined by the discussed method largely depends on the chosen ϵ vector which can drive the Pareto optimal front to both whole (the restriction did not have any effect) and empty (the restriction resulted in searching only for sub-optimal solutions), as (Deb 2014) highlights.

CHAPTER 3

Problem description and related work

In this chapter, we briefly discuss and introduce some of the most applicable related work pertaining to the subject of the thesis. All of the research utilized for this chapter is very recent and is considered cutting-edge in multi-objective combinatorial optimization.

3.1 The traveling thief problem

Combining the Travelling Salesman problem with the Knapsack problem is not a novel idea, however the relationship of their interdependence and subsequent generalization to a more substantial body of work is introduced in the work of (Bonyadi, Michalewicz, and Barone 2013). In their paper, the authors argue that despite the complexity growth of real-world problems has accelerated, researchers have continued to experiment using benchmark problems that are, in essence, similar to those used 50 years ago, having mostly substantially increased the dimensions of the problems studied.

They further indicate a gap has been observed between research and practice in metaheuristic methods, the focus of research has been mostly targeted on providing effective metaheuristics for solving well-known benchmark NP-hard problems, whereas these benchmark problems do not reflect the characteristics of real-world problems. Likewise, they elaborate that the use of the term 'complexity' as has been utilized until now, has mostly been to refer to the scale of the problems, while ignoring the actual implication of what complexity entails relating to the fundamental nature of the structure of problems. Based on that assumption, and providing some relevant examples to support this position, they extract what they believe to be two important characteristics:

- Combination - the real-world problems usually consist of two or more sub-problems that are "combined together", and

- Interdependence - in real-world problems these sub-problems are interdependent to each other in a sense that a solution for one sub-problem influences the quality of the solutions for other sub-problems.

Based on these two characteristics, they proceed to expand on their position that in complex scenarios where problems are not conforming to a standalone benchmark-style scenario, but are in fact consisting of two or more interdependent subproblems, solving one of the sub-problems in isolation, even to optimality, is not useful. Citing other studies which indicate that "many real-world problems are modelled by interdependent sub-problems. However, this aspect of interdependence is missing in the current benchmark of optimization problems". Drawing upon the facts presented up to that point, they proceed by introducing a new benchmark problem, named the Travelling Thief Problem, in an effort to provide a better approximation for real world problems, incorporating the two important characteristics itemized above, namely Combination and Interdependence.

The TTP is constructed around the TSP and KS parts, some of the characteristics of these component subproblems are summarized in the table below:

Table **3.1**: Components of the TSP and KS subproblems in the TTP.

TSP subproblem	KS subproblem
Parameters: n - the number of cities d_{ij} - the distance between city i and city j v_c - velocity	Parameters: m - the number of items w_k - weight of each item p_k - the value of each item W - total knapsack capacity
Solution representation: A tour, consisting of a permutation of the vector $\bar{x} = (x_1, x_2, ..., x_n)$ where each x_i is a city.	Solution representation: A binary (0/1) vector of picked items $\bar{y} = (y_1, y_2, ..., y_m)$ where each y_k represents whether the item was picked (1) or not (0).

For the TTP problem a similar description of the parameters and solution representation would be as in the following table:

Table **3.2**: Components of the TTP.

TTP
Parameters: Availability of each item I_i in each city (node) $A_i \in (1, ..., n)$ Problem-specific TSP/KS interdependency Solution representation: The solution to the TTP is represented by two vectors, one for the TSP which is characterized by the tour vector ($\bar{x}$) as described in the table above and one for the KS characterized by the picking plan ($\bar{z}$), a binary vector where $\bar{z} = (z_1, ..., z_m),\ z_i \in \{0 \cup A_i\} \forall$ i. By extension if $z_i = 0$ the item was not picked.

After these elements constituting the problem are introduced, two variations of the Travelling Thief Problem, TTP1 and TTP2 are presented. While each of these variations is using a different approach, they both attempt to balance the TSP and KS problems by introducing interdependency between them.

TTP1, is the first of two variations that have been introduced in this paper. In this case, two new parameters are introduced in order to make the two problems interdependent: speed and a renting rate. Speed, which is represented by $v_c = v_{max} - w_c \frac{v_{max} - v_{min}}{w}$ is a variable factor depending on the weight of the knapsack and decreases the more the knapsack is loaded to capacity. v_c stands for the current velocity, v_{max} the maximum speed, v_{min} in the minimum speed, w_c for the current weight of the knapsack and w for the maximum capacity, so as when $w_c = w$ the speed is limited to a minimum. Likewise, the renting rate is also an interdependency factor, increasing the cost by "charging" the thief for having the knapsack in their possession and is defined as a currency unit of R per unit of time. TTP1 aims to maximize the following objective function:

$$\begin{aligned} G(x, z) = g(z) - R * f(x, z) \text{ where} \\ \text{g: the total value of picked items} \\ \text{R: the renting rate per time unit in currency units} \\ \text{f: the total time of the tour, a function of both } \bar{x} \text{ and } \bar{z} \end{aligned} \tag{1}$$

Note that there have been two connections of the subproblems in this case: in the first case the *travel time* and *weight* have been connected through the utilization of **speed** as

a factor, and in the second, the *total value* $g(\bar{z})$ of the items has been connected to the *travel time* via the **renting rate R**.

TTP2, is different than TTP1, instead of minimizing or maximizing a single objective function, it tries to minimize the modified TSP and maximize the modified knapsack at the same time. Unlike in the case of TTP1, where only the TSP part was actually a multivariable function while the Knapsack subproblem was affected by the renting rate which is a constant, in this case *both* the TSP and the KS have been modified with the inclusion of new interdependency elements which turns then into multivariable functions. The new objective function for the TTP2 is as follows:

$$G(x,z) = \begin{Bmatrix} min\ f(x,z) \\ max\ g(x,z) \end{Bmatrix} \tag{2}$$

As in the case of TTP1, the functions f and g refer to the TSP and KS parts respectively.

The new elements that have been introduced in the case of TTP2 are: speed (velocity), which is defined like in the case of the previous implementation of TTP1, and a *dropping rate* which is a value depreciation factor as a function of time for the items of the Knapsack subproblem. Namely, the dropping rate is defined as $Dr\left[\frac{T_i}{c}\right]$ in the paper of (Bonyadi, Michalewicz, and Barone 2013) where T_i is the amount of time item i is carried by the thief and C is a constant. After the tour is completed, the new value of each item i in the thief's knapsack is evaluated by the function $p_i * Dr\left[\frac{T_i}{c}\right]$.

From the formulation above it is trivial to extrapolate that, like in the case of TTP1, the speed of the subject (here the thief) is affected by the weight of the picked items, hence the selections for the knapsack affect the TSP part of the problem by increasing the costs, interpreted as reduced crossing speed. The depreciation factor, implemented as a dropping rate penalty, is also influencing the value of the knapsack based on creating diminishing returns the earlier an items is selected, which in turn is also based the on position of the items in the TSP path. All other things being equal and for the same path, an item selected earlier will suffer heavier individual depreciation on its value and will simultaneously slow down the speed in which the thief crosses between nodes, whereas an item picked later will have less (average) impact on the speed and suffer less depreciation until the conclusion of the tour.

Much like in the previous case, note that there have also been two connections of the subproblems here: the *travel time* and *weight* have been connected through the utilization of **speed** as a factor, and the *value* $g(\bar{z})$ of each item has been connected to the *travel time* each has been carried for via the **dropping rate** $Dr\left[\frac{T_i}{c}\right]$.

3.2 Related work

In this section, we provide a detailed overview of the related work on the TTP. In particular, we aim at pointing out the variety of methods proposed for solving TTP which, in essence, utilize well-established methods from the domain of Combinatorial Optimization. Moreover, our main focus lies in revealing the way each research work attempts to address the interoperability of the TSP and the KS problems, since this is the real challenge the TTP brings.

3.2.1 First benchmark set and heuristics

In an attempt to advance the research on NP-hard problems that interact with one another, authors in (Polyakovskiy, Bonyadi, Wagner, Michalewicz, and Neumann 2014a) form the first benchmark set that can be used to evaluate the performance of a solver for the TTP. This benchmark set is shaped from a combination of well-known TSP and KS benchmark instances.

However, the process of finding the appropriate pairs of problem instances is not a simple one, as authors imply in the same work. That is, the combined TSP and KS instances should form a new problem instance for the TTP that well-enough balances the individual values with respect to the objective function of the TTP. Essentially, this means that the optimal solution for the respective TSP problem must not be a *good* solution for the TTP (same with the optimal KS solution). In extension, existing algorithmic implementations for the individual problem instances must be impractical for the overall TTP and, as a result, novel algorithms should be developed.

To overcome this challenge, *Polyakovskiy et. al* used as a starting point the collection of Reinelt (Reinelt 1991), i.e., the TSPLIB, for TSP instances. According to the authors, the TSPLIB can be considered the most reliable TSP problem instance collection due to (1) its variety in problem sizes, (2) nodes in instances represent real-life elements and (3) since TSPLIB is one of the oldest libraries, all problems have been exhaustively tackled to

their optimum solution. Collectively, 81 TSP instances were gathered from the TSPLIB.

Similarly, they depend on the work of (Martello, Pisinger, and Toth 1999) to select the most appropriate KS instances, all meticulously examined by *Martello et. al.* Researchers in (Martello, Pisinger, and Toth 1999) have conducted a very detailed study on what aspects of a KS instance affects the performance of a respective solver, thus generating crucial insights to *Polyakovskiy et. al* which leverage on these insights to form their final benchmark dataset. They end up with *uncorrelated*, *uncorrelated with similar weights* and *bounded strongly correlated* item types.

More specifically, for the *uncorrelated* item types, authors determine the weight and the profit values as random floats within $[1, 10^3]$. This scenario might yield easily solved KS problem instances. However, authors proceed in retaining such instances in order to investigate the behavior of the respective TTP algorithms that perform well under such instances. For the *uncorrelated with similar weights* item types, weight values are in the range of $[10^3, 10^3 + 10]$, while profits lie within $[1, 10^3]$. Last, *bounded strongly correlated* items share a weight within $[1, 10^3]$ and their respective profit is 100 more ($profit_i = weight_i + 100$, in order to enforce correlation).

In the same work, authors propose some heuristics to challenge the TTP problems instances comprising their newly formed benchmark dataset. In particular, they utilize the Chained Lin-Kernighan heuristic (Applegate, Cook, and Rohe 2003) to construct the first tour that remains untapped during the rest of the improvement procedure. Next, a Simple Heuristic (SH), a Random Local Search (RLS) and an Evolutionary Algorithm (EA) are employed to work on the KS solution. More specifically, the three methods improve the KS solution in terms of the TTP objective function and not the KS typical one. Results on some of the datasets are depicted in Tables **3.3**, **3.4** and **3.5** from (Polyakovskiy, Bonyadi, Wagner, Michalewicz, and Neumann 2014a).

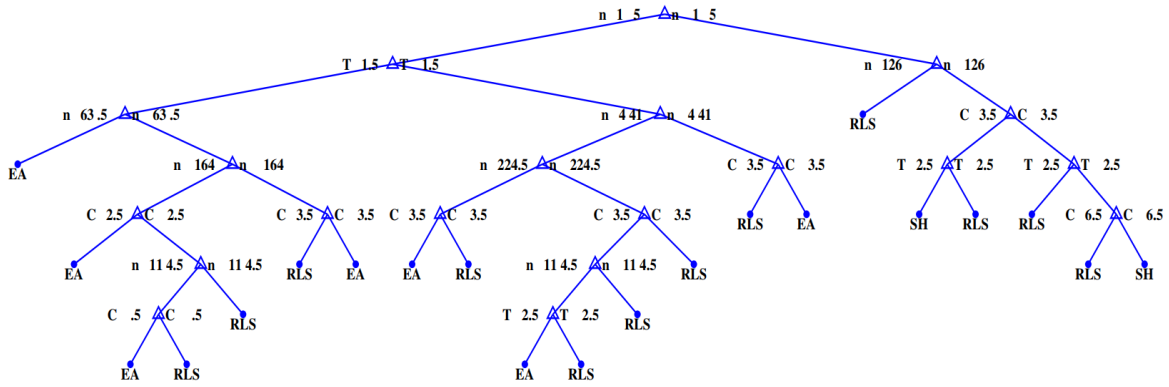

Figure **3.2.1**: The Algorithm tree for the TTP in (Polyakovskiy, Bonyadi, Wagner, Michalewicz, and Neumann 2014a) , the leaves correspond to the algorithm that produces the best result among the RLS, EA, SH

Algorithm	eil51	eil76	kroA100	u159	ts225	a280	u574	u724
RLS (mean)	8.21e3	1.16e4	1.93e4	4.03e4	5.70e4	6.32e4	1.36e5	1.67e5
RLS (std)	1.58e1	2.68e1	1.20e1	1.42e1	1.41e1	1.11e1	3.10	3.12
EA (mean)	8.22e3	1.16e4	1.93e4	4.03e4	5.70e4	6.32e4	1.36e5	1.67e5
EA (std)	1.38e1	2.49e1	9.68	1.74e1	1.01e1	1.12e1	4.42	2.72
SH	-3.03e3	-3.65e3	-4.83e3	-5.37e3	-7.49e3	-2.05e4	1.99e4	-5.16e4
PackNone	-1.46e4	-2.35e4	-2.58e4	-4.04e4	-5.57e4	-7.37e4	-1.46e5	-1.92e5

Table **3.3**: Results of RLS, EA, SH and PackNone algorithms - Part1

Algorithm	dsj1000	rl1304	fl1577	d2103	pcb3038	fnl4461	rl5934	rl11849
RLS (mean)	1.11e5	3.12e5	3.57e5	4.80e5	6.50e5	9.08e5	1.44e6	2.69e6
RLS (std)	4.10e-1	6.23e1	1.66e2	1.55e2	2.21e2	1.99e2	5.11e2	1.51e3
EA (mean)	1.11e5	3.12e5	3.57e5	4.80e5	6.50e5	9.08e5	1.43e6	2.29e6
EA (std)	2.20e1	3.30e1	1.26e1	8.92e1	5.11e1	2.03e2	1.36e4	3.46e5
SH	-1.90e5	-9.69e4	-1.30e5	-2.17e5	-2.94e5	-4.23e5	-3.97e5	-1.07e6
PackNone	-3.75e5	-3.62e5	-4.53e5	-5.78e5	-9.09e5	-1.36e6	-1.67e6	-3.45e6

Table **3.4**: Results of RLS, EA, SH and PackNone algorithms - Part2

Algorithm	d15112	d18512	pla33810	pla85900
RLS (mean)	3.44e6	3.53e6	2.02e6	-1.40e7
RLS (std)	4.96e4	1.82e5	4.61e5	6.18e5
EA (mean)	2.18e6	1.37e6	-2.16e6	-1.80e7
EA (std)	6.01e5	1.88e5	8.52e5	1.91e5
SH	-1.23e6	-1.74e6	-3.35e6	-7.49e6
PackNone	4.17e6	5.72e6	-9.28e6	-2.43e7

Table **3.5**: Results of RLS, EA, SH and PackNone algorithms - Part3

3.2.2 Investigating TSP and KS inter-dependency

The rapid growth and persistent need for versatile and efficient heuristics is proof enough for the value that can be gained by implementing fast but efficient solutions to applied problems. Real world problems, however, rarely lend themselves to the idealised and isolated academic criteria we have been utilizing in their study. Importantly, while the entire branch of heuristics is contributing significantly to dynamic and scalable problem solving, much less research effort has been dedicated to developing models that can address systems where a number of combined sub-problems interact with each other to produce a combined value function that evaluates precisely this interdependency and how it affects problems with more than one objectives.

In this section we will draw upon some observations, notes and research encountered in the bibliography about the current state of research and industry demands for complex interdependent problems and present the case made for the importance multi-objective optimization in general. Furthermore we will examine how the interdependency between the TSP and KS problems as demonstrated on the TTP acts as an indicator of the complex nature of actual industry demands and how the demand for applied problem solving is shaping the direction of future research.

In (Bonyadi, Michalewicz, Przybylek, and Wierzbicki 2014) the complexity of real-world problems is referred as a great obstacle in achieving effectiveness in contemporary business enterprises. The authors elaborate that even relatively small companies are frequently confronted in their daily operational procedures with problems of very high complexity. Several studies in real-world combinatorial optimization problems and their hardness are cited as references. However, the authors point that the reasons listed and discussed were more related to the issues of the solvers rather than the hardness of the problems themselves.Another paper is presented (Michalewicz 2012) in which real-

world problems are assigned to two broader categories: (1) design/static problems, and (2) operational/dynamic problems. The authors of the latter paper argue in part that some of the problems are first category are really hard and most of the current academic research has been concentrated on providing solutions for those problems, whereas the problems from the second category are much harder. Furthermore they also state that

> "the value of addressing the problems in the first category does not have significant influence on solving the problems in the second category".

(Bonyadi, Michalewicz, and Barone 2013) allege that

> "the sources of complexity in real-world problems in the first group is somehow different from the ones in the second group."

The argument posed in (Bonyadi, Michalewicz, Przybylek, and Wierzbicki 2014) is that real-life problems are essentially a combination of a variety of NP-Hard problems, interacting with each other, forming what they characterise as "multi-component" problems. They further argue that "current researches have been concentrated on single component problems" and "that interdependency among components in operational/dynamic problems plays a key rule in the complexity of the problems". (Bonyadi, Michalewicz, Przybylek, and Wierzbicki 2014) also present the TTP problem, a composition of two of the most famous NP-Hard and well-researched problems in the field of combinatorial optimization research, as a problem that was proposed to illustrate the complexity that arises by interdependency in multi-component problems like the TSP and KS problems that constitute its fundamental building blocks.

The authors proceed to further support their position that the real-world combinatorial optimization problems the industry is looking to solve like the Vehicle Routing problem (VRP) and Capacitate VRP (CVRP), or the KP are invariably never found as single-problem instances, and that solving the separate component problems "even to optimality" has become of less interest to the industry, leading to less interest in problem solving. They point out that apparently "the new industries are after optimization methods for multi-component problems and the methods that can only deal with single-component problems are not in interest".

3.2.3 Meta-heuristic approaches

The paper of (Wagner, Lindauer, Mısır, Nallaperuma, and Hutter 2018) at the Journal of Heuristics is a rather interesting case study of algorithms selected for approaching

the TTP problem. The paper begins by referencing that since introduction of the TTP, many algorithms have been introduced for solving it. While the initial approaches were rather generic hill-climbers, researchers incorporated more and more domain knowledge into the algorithms. As a matter of course, this resulted in deterministic, constructive heuristics, in restart strategies, and also in problem-specific hill-climbers that try to solve the TTP holistically. While the use of insights typically resulted in an increase in the objective scores, the computational complexity also increased. In their exact words "Consequently, which one of the algorithms performs best is highly dependent on the TTP instance at hand". The great number of complementarity among existing algorithms allows the authors to proceed to studying the applicability of algorithm selection to the TTP. What they put significant emphasis upon is that none of the currently proposed algorithm is significantly outperforming the others.

They continue by introducing the problem and proceeding to explain the fundamental elements that comprise the commonly accepted basic structure to what we have come to expect as the main objective from the relevant bibliography, which is also the matter being investigated in this current thesis. This will be utilised as a stepping stone in presenting the evolution of proposed algorithms for the solution of the TTP.

Polyakovskiy and his colleagues in (Polyakovskiy, Bonyadi, Wagner, Michalewicz, and Neumann 2014a) were also those who proposed the first set of heuristics for solving the TTP. Their general approach was to solve the problem using two steps: The first step involved generating a "good" TSP tour by using the classical Chained Lin-Kernighan heuristic, a very well evaluated construction algorithm generally considered to be a reference point as a construction heuristic algorithm for the symmetric TSP. The second step involved keeping the tour immutable while applying a packing heuristic for improving the solution. Their first approach was with the use of simple heuristic (SH) which constructs solutions by processing and picking items that maximize the objective value according to a given tour. In this specific case Items were picked based on a score value that was calculated for each item to estimate how good it was according to the tour provided. They further proposed two iterative heuristics, namely the Random Local Search (RLS) and (1+1)-EA, which probabilistically flipped a number of packing bits. After each iteration the solution was evaluated and if an improvement was noted, the changes were accepted and maintained, otherwise they were ignored.

After presenting the work of the original authors, they proceed to the next paper that was presented in chronological order and was investigating the TTP, which is the work

of (Bonyadi, Michalewicz, Przybylek, and Wierzbicki 2014). In this paper the authors experimentally investigated the interdependency between the TSP and knapsack components of the TTP. They proposed two heuristic approaches named Density-based Heuristic (DH) and CoSolver. DH is once more a two-phased approach similar to the SH that had been utilized from Polyakovskiy, and it too ignores any dependencies between the TSP and Knapsack components. The second approach however utilizes CoSolver which is very different. CoSolver is "a method inspired by coevolution based approaches": The way it operates is that it divides the problem into sub-problems where each sub-problem is solved by a different module of the CoSolver. The algorithm revises the solution through "negotiation" between the incorporated modules. The communication between the different modules and sub-problems gives a means for the TTP interdependencies as defined by (Polyakovskiy, Bonyadi, Wagner, Michalewicz, and Neumann 2014a) to be considered. A comparison across several benchmark problems demonstrated the superiority of their second approach, CoSolver, over DH. This was rather more demonstrative in the case of larger instances. A simplified graph for the CoSolver framework for the TTP from (El Yafrani and Ahiod 2018) can be seen in figure **3.2.2**.

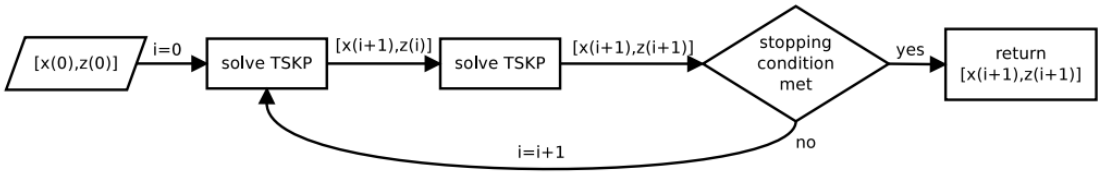

Figure **3.2.2**: A simplified CoSolver framework for the TTP sourced from (El Yafrani and Ahiod 2018)

In 2016, Mei et al. in (Mei, Li, and Yao 2016) intrigued by the concepts of combination and interdependance that had been introduced by (Bonyadi, Michalewicz, and Barone 2013), also investigated the interdependencies between the TSP and knapsack components. Initially they analysed the mathematical formulation to show that the TTP problem is not additively separable into individual unrelated subcomponents. Demonstrating that since the objectives of the TSP and knapsack components are not fully correlated, one cannot expect to achieve competitive results by solving each component in isolation. The authors used two separate approaches for solving the TTP: a cooperative coevolution based approach (similar to CoSolver), and a memetic algorithm called MATLS which attempts to solve the problem as a whole. The memetic algorithm, which

considered the interdependencies in more depth, outperformed the cooperative coevolution results of the CoSolver that had been published by Bonyadi et al. two years prior. Below are tables demonstrating the advantage of MATLS over CoSolver (and hence the original SH and Random Local Search approaches) as found in (Mei, Li, and Yao 2016).

A little before the publication of (Mei, Li, and Yao 2016) however, Faulkner et al. in (Faulkner, Polyakovskiy, Schultz, and Wagner 2015), investigated multiple heuristic search operators and did a comprehensive comparison with existing approaches. They also proposed a number of operators, such as Bitflip and PackIterative, for optimising the Knapsack packing plan given a particular tour and also proposed Insertion for iteratively optimising the tour given a particular packing. They combined these operators in a number of simple (S1-S5) and complex (C1-C6) heuristics that outperformed existing approaches. The main observation of their research was that there did not yet seem to be a single best algorithmic paradigm for the TTP. According to Wagner et al. (Wagner, Lindauer, Mısır, Nallaperuma, and Hutter 2018) the individual operators were quite beneficial in improving the quality of results. Wagner however also stresses that while the operators that were proposed in that paper seemed to have certain benefits, the simple and complex heuristics did not consider the interdependencies between the TTP components, since all of these approaches were multi-step heuristics. The best approach that the paper from (Faulkner, Polyakovskiy, Schultz, and Wagner 2015) seemed to find was, surprisingly according to Wagner, a rather simple restart approach name S5 that combined good TSP tours with the fast PackIterative heuristic search operator.

A different approach was followed by of (El Yafrani and Ahiod 2018), they proposed two algorithms, namely; CS2SA* and CS2SA-R, an approach combining Hill Climbing and Simulated Annealing, implementing the CoSolver framework. In their approach, the initial tour is generated using the Concorde's implementation of the Lin-Kernighan heuristic, while the picking plan is initialized using The insertion heuristic proposed in (Mei, Li, and Yao 2014). After applying the insertion heuristic, they utilize a bit-flip search on the generated knapsack to eliminate some items considered to be of low quality. To further improve the outcome, they also utilize some performance enhancement techniques: TSKP neighborhood reduction, the Delaunay triangulation as a candidate generator for the 2-OPT, as well as Objective value recovery where they keep track of time and weight information at each city of a given tour to accelerate the recovery of objective value without using the objective function to evaluate neighbors. For larger instances, according to the authors "2-OPT and the bit-flip search take too long" and

2-OPT with early break and insertion without elimination were used utilized instead. The instances used to evaluate the performance of CS2SA* and CS2SA-R in (El Yafrani and Ahiod 2018) were split into three categories with 1, 5 and 10 items respectively that also shared similar properties as pertaining to the correlation between their weights and values:

- Category 1 (F=1, T=3, C=1) : 1 item per city, item values and weights are bounded and strongly correlated, small knapsack capacity.
- Category 2 (F=5, T=2, C=5) : 5 items per city, KP uncorrelated but items have similar weights, average knapsack capacity.
- Category 3 (F=10, T=1, C=10) : 10 items per city, KP uncorrelated, high knapsack capacity.

instance	MATLS			S5			CS2SA*			CS2SA-R		
	mean	**rsd**	**time**	**mean**	**rsd**	**time**	**mean**	**rsd**	**time**	**mean**	**rsd**	**time**
eil76	22,188	0.78	3.2	20,097	0	600	18,753	0	231	22,032	1.2	600
kroA100	42,595	1.31	6.27	39,440	0	600	39,271	0	54	43,712	2.42	600
ch130	61,061	0.11	75	58,708	1.27	600	50,695	0	103	60,864	0.72	600
u159	58,289	1.33	10	57,618	0	600	58,090	0	123	60,377	1.03	600
a280	110,132	2.34	31	109,921	0.01	600	107,696	0	196	114,087	0.84	600
u574	254,770	0.71	222	251,775	0.05	600	248,584	0	116	248,402	1.05	600
u724	303,435	1.17	193	305,977	0.35	600	309,636	0	63	303,025	1.11	600
dsj1000	340,807	1.55	383	342,189	0.59	599	332,883	0	109	339,136	2.21	600
rl1304	572,766	1.18	290	575,102	0.86	600	585,600	0	181	578,064	1.41	600
fl1577	609,288	1.77	442	607,247	1.62	598	636,425	0	281	584,764	3.23	600
d2103	849,632	1.28	495	853,587	1.18	600	842,520	0	301	849,770	2.9	600
pcb3038	1,168,108	0.45	581	1,179,510	0.2	597	1,193,738	0	365	1,159,828	0.9	600
fnl4461	1,617,401	0.3	600	1,625,856	0.18	596	1,628,414	0	157	1,588,086	0.64	600
pla7397	4,178,551	3.87	600	4,371,424	0.79	591	3,713,314	0	383	3,798,791	9.37	600
rl11849	4,587,848	0.41	600	4,630,753	0.29	587	4,710,135	0	600	4,668,813	1.22	600
usa13509	7,767,305	2.02	600	7,818,124	0.72	579	8,115,168	0	600	7,930,181	2.54	600
brd14051	6,492,947	1.13	600	6,552,858	0.61	587	6,654,162	0	600	6,667,470	1.09	600
d15112	6,828,152	2	589	6,991,440	1.31	578	7,606,856	0	600	7,561,526	3.18	600
d18512	7,164,421	1.27	581	7,257,709	0.68	600	7,579,996	0.01	600	7,324,264	1.98	600
pla33810	15,532,990	1.1	600	15,574,552	0.73	566	15,704,051	0.5	600	14,421,765	451	600

Table **3.6**: An excerpt from the results of CS2SA* and CS2SA-R in (El Yafrani and Ahiod 2018)

3.2.4 Population Based vs Single Point Search

Yafrani and Alhiod (El Yafrani and Ahiod 2016) quite accurately mention that the TTP, despite being an artificial problem, can serve as a benchmark problem for the investigation of inter-dependency between sub-problems. According to the authors, promoting research in this direction, *i.e.*, studying the inter-dependency of problems, can narrow down the distance that separates real-life problems and theoretical ones. To this end, they examine the behavior of what they conceive as the state-of-the-are methods for the TTP, namely a memetic algorithm (*MATLS*) and the *PACKITERATIVE* heuristic. Despite both of them are not devised by (El Yafrani and Ahiod 2016), we consider that this is a good opportunity to briefly discuss their search mechanisms for completeness reasons.

The *MATLS* algorithm is driven by four core elements. The *population initialization* phase, where 30 solutions are generated. The tours of the first 10 individuals are obtained by the Chained Lin-Kernighan Heuristic (Applegate, Cook, and Rohe 2003) and a Minimum Spanning Tree for the remaining 20, while a 2-OPT local search follows that shapes the final tour sub-solutions. The picking plan is generated using a greedy algorithm that uses a goodness score to sort items. For further information about the greedy algorithm, authors point to (Mei, Li, and Yao 2014). The *crossover operator* used is the Order Crossover (Goldberg, Lingle, et al. 1985) and is only applied upon the tour sub-solution. The *Inner Local Search* is the same as in the initialization step, however it here happens with a small probability. A child solution is kept and replaces an offspring solution if and only if the former scores better than the latter. Last, the *stopping criterion* of the algorithm is set to 10 minutes. Figure **3.2.3** illustrates the process the *MATLS* follows to obtain the best solution.

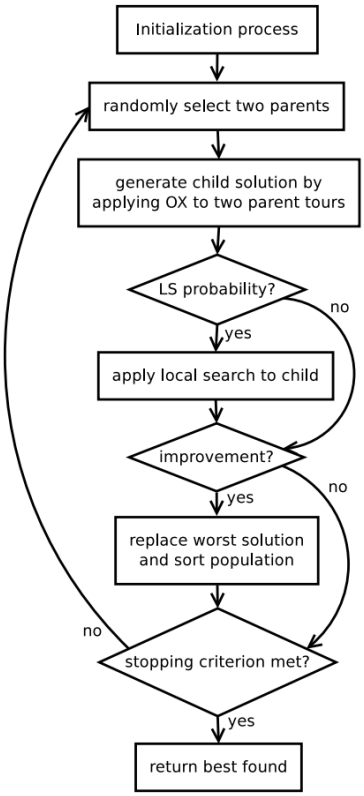

Figure **3.2.3**: The MATLS tree as illustrated in (El Yafrani and Ahiod 2016)

The *PACKITERATIVE* heuristic (for short, S5) was initially introduced in (Faulkner, Polyakovskiy, Schultz, and Wagner 2015). It is a single-point search heuristic that forms the initial tour, once again, with the Chained Lin-Kernighan Heuristic. In addition, it utilizes an iterative greedy algorithm that determines the packing plan. More specifically, each item is assigned a score that depends on its profit, weight and distance from the last city of the tour, while the profit and weight are strengthened using an exponent, which are varied iteratively. As for the stopping criterion, it is set to 10 minutes of exploration time.

Before diving into the new methodologies proposed in (El Yafrani and Ahiod 2016), we should point out a short discussion authors pose about ways of evaluating neighboring solutions in the TTP. Since the latter is proved a computationally extremely expensive problem, the computation of an objective value is quite time consuming. Therefore, some techniques are usually applied to reduce the search space or to speed up the computation of a neighboring solution. For instance, in *MATLS*, information of the initial solution's objective value is used to calculate a neighboring one, while in *S5* the calls to the objective

function occur based on some frequency value, skipping consecutive evaluations.

The first algorithmic proposal of *Yafrani and Alhiod* is a single-point search algorithm that they name *CS2SA*. Essentially, *CS2SA* forms an initial solution of the TSP with the Chained Lin-Kernighan Heuristic and, for the picking plan, they utilize a similar heuristic as in (Mei, Li, and Yao 2014). Concerning the improvement of the initial obtained TSP solution, a neighborhood is defined under the 2-OPT neighborhood operator and the best solution is chosen. We should note that, authors deploy Delaunay triangulation in order to reduce the search space. The KS sub-solution is improved with an appropriate Simulated Annealing metaheuristic utilizing the bit-flip neighborhood operator. An illustration of the steps CS2SA follows are given in Figure **3.2.4**.

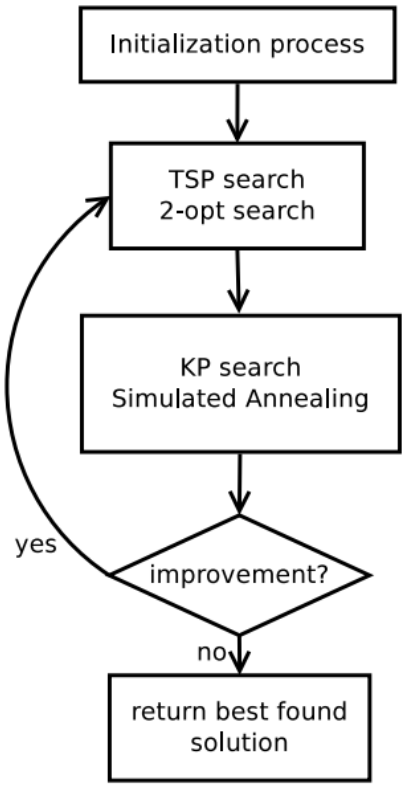

Figure **3.2.4**: The modified Simulated Annealing (CS2SA) tree from (El Yafrani and Ahiod 2016)

For a population-based TTP solver, authors implement a memetic algorithm called *MA2B*. For brevity reasons, we will describe only the key-concepts of this algorithm. According to the authors, the initial tour plays a crucial part in the search process of a

TTP algorithm. Therefore, for the initial TTP solution, they utilize the Chained Lin-Kernighan Heuristic only for a limited amount of iterations, they apply a greedy heuristic for a picking plan and, finally, they apply a Restrictive Local Search algorithm that uses 2-OPT and bit-flip to improve the tour and the packing plan, respectively. This process is restricted to only 50 iterations, since in population-based algorithms local search is considered very expensive. Last, two parent solutions are selected with the tournament technique, while the Maximal Preservative Crossover (Mühlenbein 1991) is utilized as the crossover operator.

The four methods described above were tested on a multitude of TTP problem instances. The results are shown in Table **3.7**.

Instance	MATLS			S5			CS2SA			MA2B		
	objective	**rsd**	**time**	**objective**	**rsd**	**time**	**objective**	**rsd**	**time**	**objective**	**rsd**	**time**
eil76	3,705	1.38	4.8	3,742	0	600	3,906	3.49	5.36	4,290	2.46	61
kroA100	4,659	1.34	67	4,278	0	600	4,464	0.01	7.89	4,575	2.69	63
ch130	8,876	0.83	62	9,250	0	600	9,084	0.32	14	9,356	0.34	109
u159	8,403	1.42	3.07	8,634	0	600	8,436	0.63	23	8,646	0.16	186
a280	17,678	0.54	7.17	18,409	0	600	18,467	0.06	45	18,294	0.74	465
u574	26,121	2.15	193	26,923	0.17	600	26,440	1.37	19	26,648	1.18	600
u724	48,967	1.25	44	50,310	0.19	600	49,754	0.35	37	49,224	0.66	600
dsj1000	143,699	0	44	137,654	0.16	598	144,219	0	22	144,219	0	600
rl1304	75,800	1.26	63	79,710	0.56	600	75,134	0.57	57	77,942	0.47	600
fl1577	88,375	0.41	142	92,169	1.25	597	90,149	0.4	93	93,742	0.47	600
d2103	113,005	0.47	184	120,563	0.18	600	120,682	0.01	527	119,908	0.69	600
pcb3038	148,265	1.14	330	159,929	0.18	598	150,499	0.46	368	157,804	1.19	600
fnl4461	247,553	0.37	479	262,367	0.12	596	248,861	0.24	600	256,797	0.37	600
pla7397	365,506	1.11	578	395,595	0.55	591	314,838	0.02	600	368,985	0.82	600
rl11849	661,283	0.29	600	708,388	0.21	591	658,882	0.4	600	680,412	0.34	600
usa13509	747,905	0.47	600	809,543	0.33	582	684,424	0.15	600	772,507	0.62	600
brd14051	815,511	0.37	600	876,190	0.09	586	818,939	0.2	600	856,632	0.04	600
d15112	871,069	0.52	592	939,389	0.26	585	872,575	0.11	600	914,056	0.62	600
d18512	996,544	0.77	600	1,073,230	0.18	582	1,007,065	0.15	600	1,048,409	0.18	600
pla33810	1,730,207	0.87	600	1,872,169	0.97	572	1,775,854	0.14	600	1,763,677	0.59	600

Table **3.7**: An excerpt from the results of in (El Yafrani and Ahiod 2016) (category 2)

CHAPTER 4

Methodology

This chapter accommodates the procedure we followed in order to engage the TTP problem. We start by presenting the notion of a solution representation as well as the definition of a fitness function. Afterwards, we determine the composition of the neighborhood structures, an integral part of every VNS algorithm, and introduce construction heuristics that improve the performance of our algorithm. Finally, we substantiate the selection of some core parameters used in the frame of our work, e.g., termination criteria.

4.1 Solution representation

As considered in Chapter 3, the TTP concerns a combination of the TSP and the KS problems. Therefore, an intuitive representation of a TTP solution would be the aggregation of the typical TSP solution representation, i.e., an array consisting of a permutation of n different non-negative integers, and the characteristic KS solution representation, i.e., a binary array with ones denoting the selection of an item and zeros the opposite.

Example 1. Consider a TTP with four nodes (cities) and let city 1 be the starting node. Each node is associated with a single item, except for the first. Let us also denote with s a random solution of the TTP. Then:

$$s = \{s1, s2\} \text{ with } s_1 = [1, 2, 4, 3] \text{ and } s_2 = [0, 1, 0, 1]$$

is interpreted as the traveling thief following the route $1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$ and picking the items of cities 2 and 4.

The example above is indicative and aims to illustrate the extreme computational complexity of the TTP problem. Even in this simple scenario consisting of four nodes

and three items (since, by definition, no item is assigned to the first city), the solution space includes

$$3! \times 2^3 = 48$$

possible permutations.

4.2 Objective function

An appropriate evaluation function should consider both the route traversed by the traveling thief and the value of the items reside in the knapsack. The special part about the TTP, however, is that a heavy knapsack slows down the thief. That is, the knapsack weight directly affects the cost (e.g., time) of each edge between two cities. Formally, the cost $c_{i,j}$ of the edge $i \rightarrow j$ should be conceived as a function of w_i which denotes the weight of the knapsack being carried up to city i. This peculiarity demonstrates the need for a specialized objective function for the TTP.

Polyakovskiy et.al (Polyakovskiy, Bonyadi, Wagner, Michalewicz, and Neumann 2014a) within their work, introduced an appropriate objective function that considers all of these preceding parameters. Let $N = \{1, ..., n\}$ be the set of cities and $d_{ij}, i, j \in N$, the given distance between any pair of cities. Each city i, except for the first one, is assigned a set of items $M_i = \{1, ..., m_i\}$ and, furthermore, each item k within city i is assigned the properties of value p_{ik} and weight w_{ik}. Then, Equation 1 can be used as our objective function.

$$maximize \quad Z(\Pi, P) = \sum_{i=1}^{n} \sum_{k=1}^{m_i} p_{ik} y_{ik} - R(\frac{d_{x_n x_1}}{u_{max} - vW_{x_n}} + \sum_{i=1}^{n-1} \frac{d_{x_i x_{i+1}}}{u_{max} - vW_{x_i}}) \qquad (1)$$

where:

Π = the tour, i.e., $\{x_1, ..., x_n\}, x_i \in N$
P = the packing plan, i.e., $\{y_{21}, ..., y_{nm_i}\}$
y_{ik} = binary variable, selection of item k from city i
R = the renting rate paid each time unit being on a way
v = $v_{max} - v_{min}$
W_i = the knapsack weight when leaving city i
W = the maximum knapsack's weight

Quoting from the same work:

> *"The minuend is the sum over all packed items' profits and the subtrahend is the amount that the thief pays for the knapsack's rent equal to the total traveling time along* Π *multiplied by R."*

The apparent objective is to minimize Equation 1.

4.2.1 Illustrative example

Assume a TTP comprised of four nodes (cities) comprised of one item per node, the route $1 \rightarrow 4 \rightarrow 2 \rightarrow 3 \rightarrow 1$ and the selection of items in cities 3 and 4, as demonstrated in Figure **4.2.1**. Moreover, consider the Tables **4.1**, **4.2** and **4.3** depicting the input of the problem.

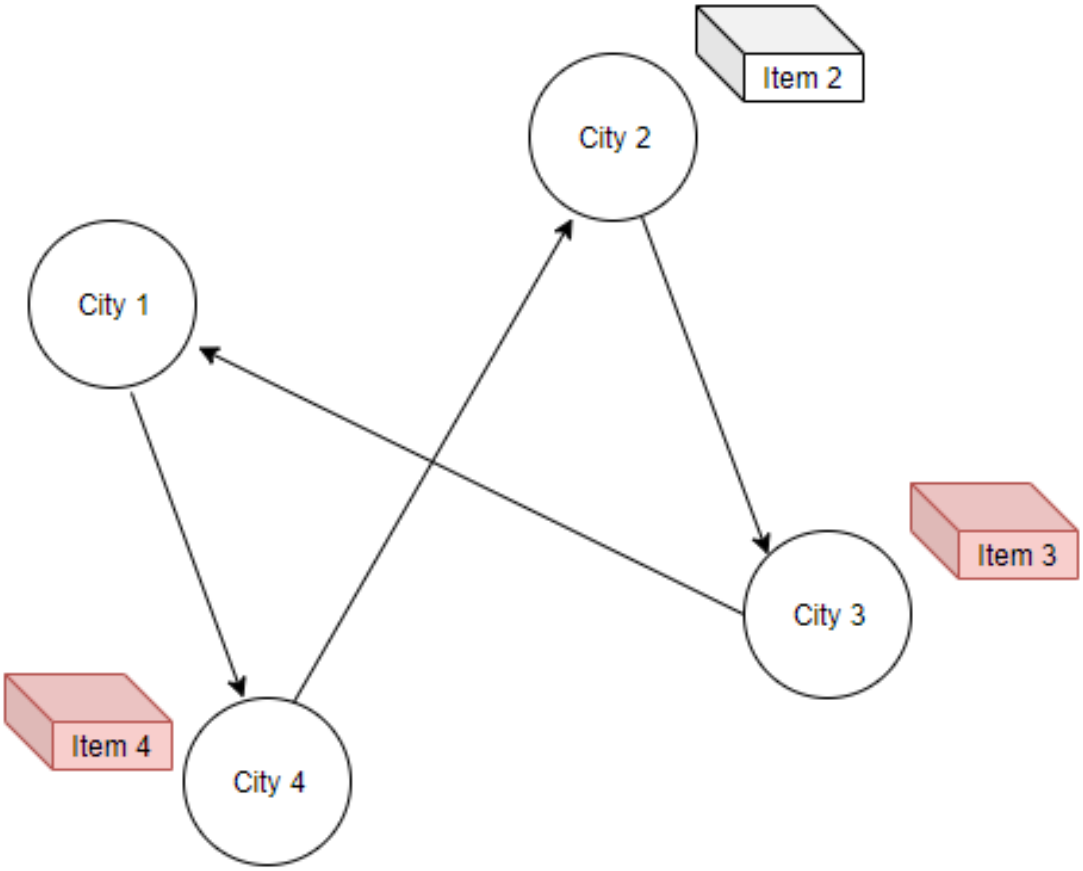

Figure **4.2.1**: Illustration of a TTP solution.

Table **4.1**: The input data concerning the TTP environment.

Parameter	**Value**
R	1
v_{max}	1
v_{min}	0.1
W	12

Table **4.2**: The input data concerning distances between the cities.

City	**coordinate x**	**coordinate y**
1	0	0
2	2	1
3	1	-3
4	4	-2

Table **4.3**: The input data concerning the attributes of the items.

Item	weight	value
2	8	5
3	4	3
4	2	3

Then, the solution:

$$s = \{s_1, s_2\}, s_1 = \Pi = \{1, 4, 2, 3, 1\}, s_2 = P = \{0, 1, 1\}$$

described above will yield:

$$\begin{aligned} Z(\Pi, P) = Z(\{1,4,2,3,1\}, \{0,1,1\}) = \\ = \sum_{i=1}^{4}\sum_{k=1}^{1} p_{ik}y_{ik} - 1(\frac{d_{4,1}}{1-0.9\cdot 6} + \sum_{i=1}^{3}\frac{d_{x_i x_{i+1}}}{1-0.9\cdot W_{x_i}}) = \\ \sum_{i=1}^{4}\sum_{k=1}^{1} p_{ik}y_{ik} - 1(\frac{d_{4,1}}{1-0.9\cdot 6} + \frac{d_{x_1x_2}}{1-0.9\cdot W_{x_1}} + \frac{d_{x_2x_3}}{1-0.9\cdot W_{x_2}} + \frac{d_{x_3x_4}}{1-0.9\cdot W_{x_4}}) = \\ ... = -9.724223572153782 \end{aligned} \tag{2}$$

4.3 Neighborhood definition

In Chapter 2 we adduce how VNS algorithms utilize multiple neighborhood structures to improve upon an existing solution. In this section, we describe the two neighborhoods we implement to tackle the TTP.

First neighborhood Given a solution s, the set of $N_1(s)$ contains every solution s' that has one item flipped from the existing packing plan (s_2). For instance, if $s = \{[1, 2, 3, 4], [0, 0, 1, 0]\}$, then $s' = \{[1, 2, 3, 4], [0, 1, 1, 0]\} \in N_1(s)$.

Second neighborhood Given a solution s, the set of $N_2(s)$ contains every solution s' that has one selected item removed from a city at the beginning of the route and one item from a city added at the end of the route. Cities pertaining to the first half of the route are considered to be at the beginning while the rest are considered to be at the end. For instance, if $s = \{[1, 2, 3, 4, 5], [0, 1, 1, 0, 0]\}$, then $s' = \{[1, 2, 3, 4, 5], [0, 0, 1, 0, 1]\} \in N_2(s)$.

According to the neighborhood definitions above, we can sufficiently explore and evaluate a wide variety of solutions.

4.4 Solution initialization

During our experimentation, two separate construction methods are used, a typical *greedy* and a *custom greedy* approach.

Greedy The greedy procedure utilizes some heuristic criteria to construct an initial solution. More specifically for the permutation of the nodes (cities), for every city i it selects the next city j, with j being city the closest to i.

Item-wise, we rank items with respect to their $r = \frac{value}{weight}$ ratio. We ostensibly want to favor items with high r. After sorting items in descending order according to their r-value we sequentially fill the knapsack with the highest rated of them until it cannot accommodate any more items.

Custom Greedy The custom greedy construction heuristic utilizes an array of more sophisticated criteria compared to the greedy heuristic. Although it initiates with a permutation of cities structured by the nearest neighbor rule, our own greedy algorithm implementation that incorporates the *risc factor* associates the position of each city in the permutation to assign a risk factor of selecting a particular item in conjunction with its r value, as described in the previous paragraph, in an **inverted** way.

The risk factor value of selecting item i is calculated as:

$$risk_i = (1 + p_s) \cdot \frac{1}{r_i}$$

where p_s denotes the number of outstanding cities to reach at the last city of the permutation, counting from city s. For instance, consider the city permutation $s_1 = [1, 3, 4, 2]$ and the item ratios vector $r = [1, \frac{1}{2}, \frac{1}{3}]$. Then,

$$risk_3 = (1 + p_3) \cdot \frac{1}{\frac{1}{2}} = (1 + 2) \cdot 2 = 6$$

We implement both solution initialization techniques described above in order to examine the extent of which our proposed VNS and RVNS algorithms can improve the given solu-

tions. In Chapter 5 we present all of our findings including tables and graphs regarding these performance measurements.

CHAPTER 5

Results and Comparison

This chapter assesses the performance of the methodologies proposed, on the Travelling Thief problem. The tables containing the results follow the presentation format of the algorithmic approaches in (Polyakovskiy, Bonyadi, Wagner, Michalewicz, and Neumann 2014a), (Mei, Li, and Yao 2016), (El Yafrani and Ahiod 2016), (El Yafrani and Ahiod 2017), (El Yafrani and Ahiod 2018) and (Wu, Wagner, Polyakovskiy, and Neumann 2017) while for the presentation of the graphs of the larger datasets we opted to utilize the approach followed by (Mei, Li, and Yao 2016) which presents the evolution of the value of the objective function on the y axis and the algorithm runtime in the x axis. We further incorporate the results for both algorithms which were implemented; namely RVNS and VNS, for each of the two construction function heuristics, greedy1 and greedy2, in the same graphs. This is to facilitate the comparison between the two methods and the effect of the construction functions in enhancing both the value of the final outcome, as well as the time required to achieve it.

Results varied significantly depending on the size of the dataset, the solutions for larger datasets demonstrating notable deviation from the optimal solutions as presented in the relevant literature referenced in the previous paragraph. This can be attributed to the authors of these algorithms utilizing methods to reduce the size of the solution space and improve efficiency. Such techniques, like Delaunay triangulation in both (El Yafrani and Ahiod 2016) and (El Yafrani and Ahiod 2018) and TSKP neighborhood reduction and Objective value recovery on the latter, were referenced in Chapter 3. In this work we separate the datasets that will be examined into two subcategories, presenting the results in the smaller datasets in more detail.

- Category 1 - contains the results obtained in the smaller eil76, kroA100, ch130 and u159 instances comparing them to the corresponding best values presented in

(El Yafrani and Ahiod 2018) under their category 1 classification, with 1 item per city, item values and weights bounded and strongly correlated as well as a small knapsack capacity. The specifics of each dataset are presented in Table **5.1**

- Category 2 - includes two larger datasets, a280 and fnl4461, with variations in the number of items (i.e. 1, 5 and 10) that significantly raises complexity. The performance of both VNS and RVNS in this category is not efficient and we have instead opted to utilize it mostly to demonstrate the differences between our methods rather than as a competitive approach to the TTP. The exact composition of the variations on these problems is presented in Table **5.2**. For reasons of clarity and brevity we label the datasets (see **Id** column). The datasets are sourced from the IEEE CEC 2014 Competition as described in (Polyakovskiy, Bonyadi, Wagner, Michalewicz, and Neumann 2014b).

To further document the specifics of our RVNS implementation, we have opted to generate and include figures that present the absolute number of improvements yielded from each of the two neighborhoods for each of the construction heuristics and methods in both categories. This allows us to observe how the selection of neighborhoods in this particular local search context contributes to the solutions generated.

Id	**Cities**	**Items per city**	**Correlation Weight/Profit**
eli76	76	1	strongly correlated
kroA100	100	1	strongly correlated
ch130	130	1	strongly correlated
u159	159	1	strongly correlated

Table **5.1**: Category 1 problem instances. The datasets used are sourced from the *IEEE CEC 2014 Competition website*

Id	**Cities**	**Items per city**	**Correlation Weight/Profit**
A1	280	1	strongly correlated
B1	280	5	uncorrelated but similar
C1	280	10	uncorrelated
A2	4461	1	strongly correlated
B2	4461	5	uncorrelated but similar
C2	4461	10	uncorrelated

Table **5.2**: Category 2 problem instances. The datasets used are sourced from the *Adelaide competition.*

It is worth noting that, the last column in Table **5.2** introduces information that might prove to be of considerable importance from a performance standpoint, since it has a significant effect on the quality of the solutions yielded by the construction heuristics we deploy. That is, strongly correlated weight and profit values (per item), which essentially translates to that the more valuable an item is the more it weighs. Hence, by applying the greedy1 heuristic to acquire the knapsack solution initialization, we are dealing with a limited range of results in profit/weight ratios and, therefore, the policy we follow is not exploited to its full potential. In the other two cases though, things are more favourable for the greedy1 heuristic since ratios span both ranges (0,1) and [1,..).

5.1 Performance Evaluation - Category 1

5.1.1 Performance on the eli76 dataset

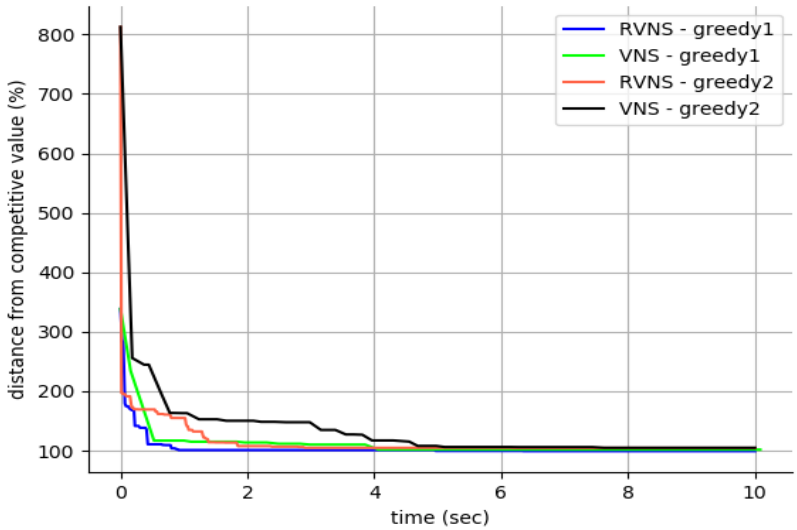

Figure **5.1.1**: Performance of all methods on the eli76 dataset.

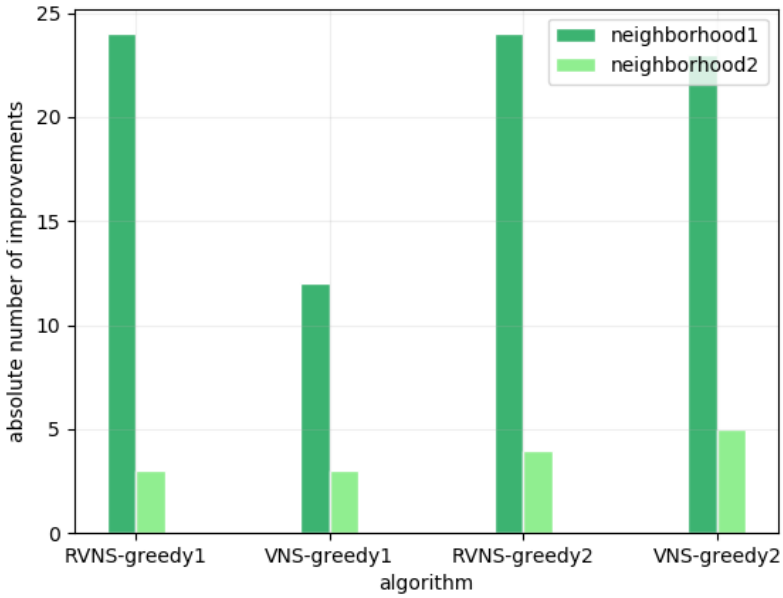

Figure **5.1.2**: Improvements per neighborhood yielded from all methods on the eli76 dataset.

From Figure **5.1.1** we can observe that by $t = 5$ seconds all four algorithms have converged. The improvement over the double span of $t = 10$ seconds is negligible. This demonstrates rapid convergence that falls, however, far from the known competitive solution cited in (El Yafrani and Ahiod 2018) found with the CS2SA-R algorithm. RVNS-greedy1 converges faster than all methods, in under $t = 1$ second, while VNS-greedy2 converges last, in around $t = 5$ seconds. It will become clear below that this seems to be a consistent trend. Figure **5.1.2** demonstrates all neighborhoods contribute to the solution with the majority of contributions, however, coming from the first neighborhood.

5.1.2 Performance on the kroA100 dataset

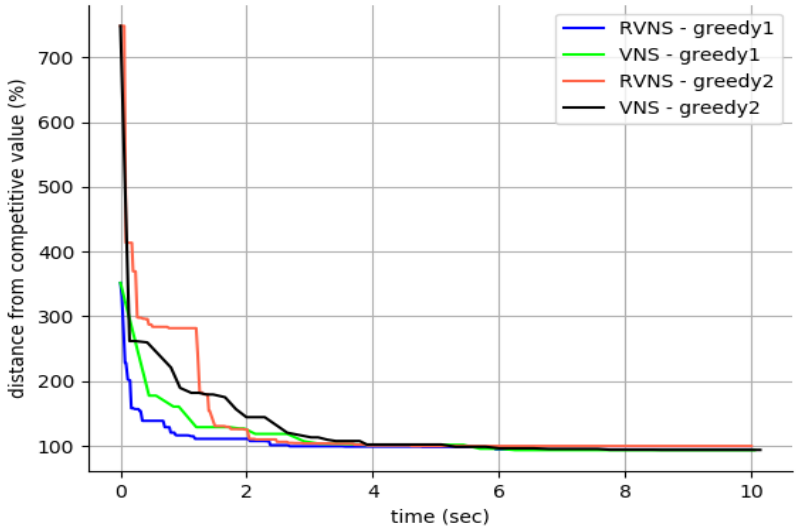

Figure **5.1.3**: Performance of all methods on the kroA100 dataset.

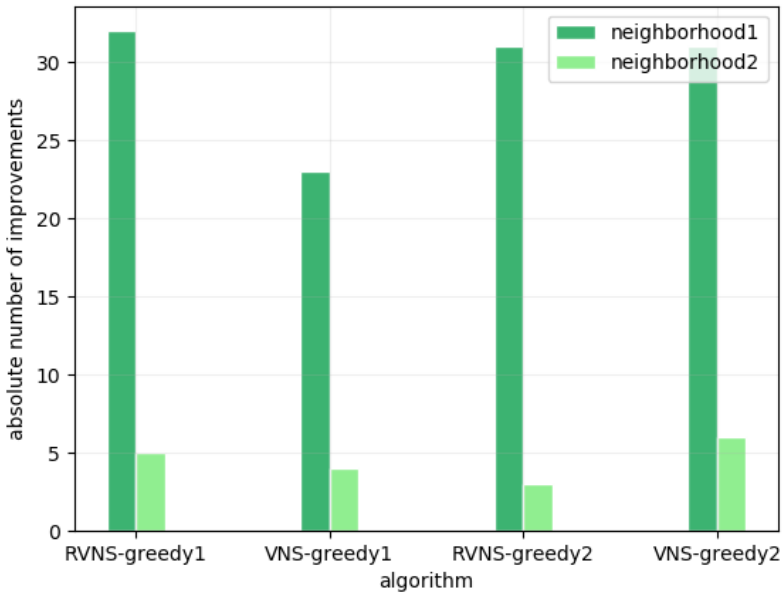

Figure **5.1.4**: Improvements per neighborhood yielded from all methods on the kroA100 dataset.

The bigger dataset used here does not seem to make the convergence of our algorithms slower. Figure **5.1.3** shows rapid convergence among all methods up to $t = 4$ seconds and a continued improvement at a much slower rate afterwards. Once again RVNS-greedy1 converges faster than all methods just after $t = 1$ second. RVNS-greedy2 starts slowly but after $t = 2$ seconds it surpasses both VNSs. Once more VNS-greedy2 is the last one to reach convergence. The contributions from neighborhood 2 however seem to have declined compared to the *eli76* dataset across all algorithms but especially in the case of RVNS-greedy2.

5.1.3 Performance on the ch130 dataset

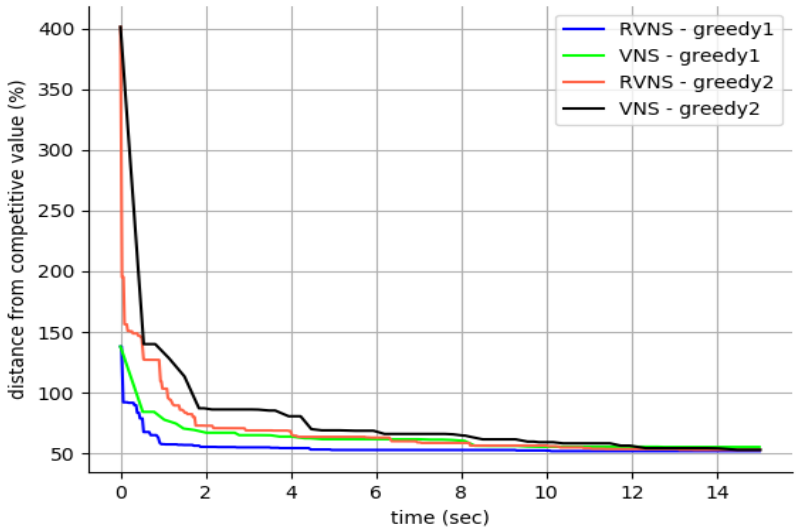

Figure **5.1.5**: Performance of all methods on the ch130 dataset.

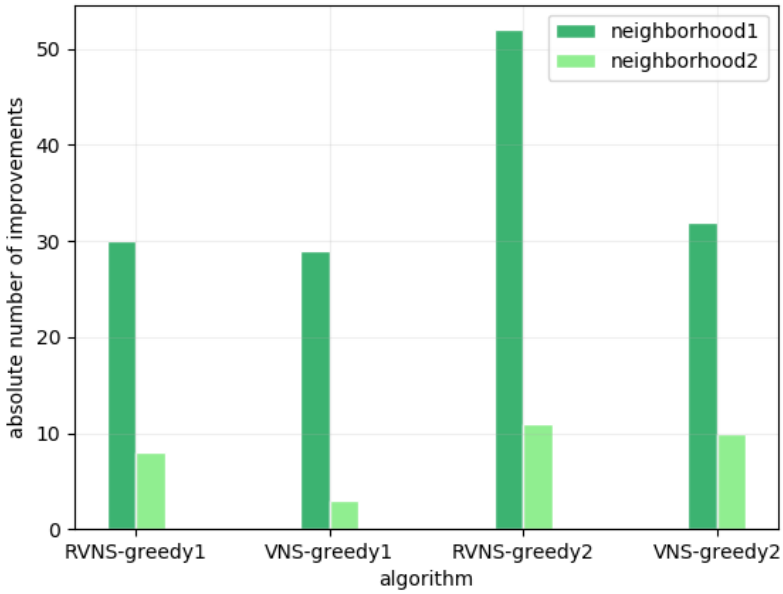

Figure **5.1.6**: Improvements per neighborhood yielded from all methods on the ch130 dataset.

With 130 nodes, complexity has increased significantly and the longer time to visible convergence in Figure **5.1.5** demonstrates that. Ultimately convergence between all four implementations seems to be achieved between $t = 13$ and $t = 14$ seconds. RVNS-greedy1 once again seems to be the fastest, achieving the majority of the improvement under $t = 1$ second, despite the longer runtime until all methods begin to reach an equilibrium of minimal improvement for every unit of time. VNS-greedy2 is once again the slowest of the three. Figure **5.1.6** indicates that is the first test where neighborhood 2 appears to make significant contributions, reaching up to almost 30% of the total neighborhood contributions in some cases.

5.1.4 Performance on the u159 dataset

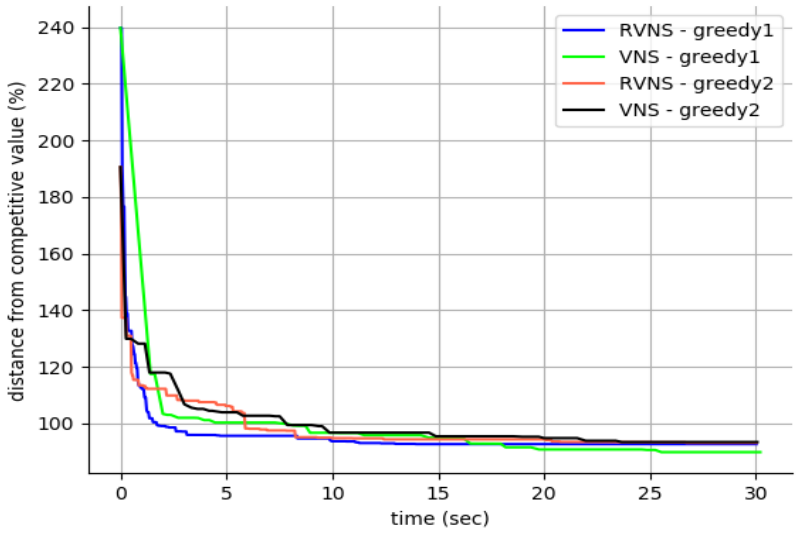

Figure **5.1.7**: Performance of all methods on the u159 dataset.

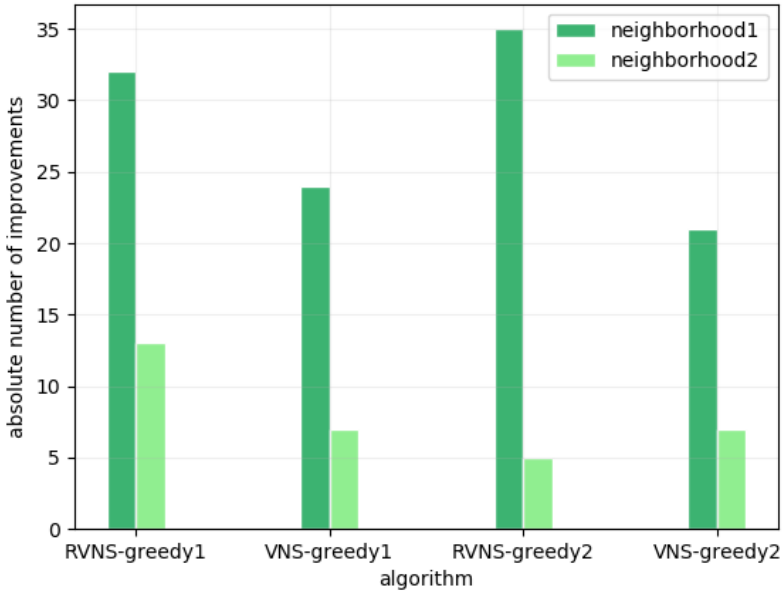

Figure **5.1.8**: Improvements per neighborhood yielded from all methods on the u159 dataset.

In this last of the smaller datasets, the increase in the complexity caused by the introduction of more nodes and items significantly widens the timespan required until we can observe a relative convergence bewteen the methods. For a fourth consecutive time, RVNS-greedy1 demonstrates itself to be the fastest of the methods reaching a threshold of minimal improvement after between $t = 3$ and $t = 4$ seconds, outperforming all the others. Interestingly, while VNS-greedy1 begins with the slowest convergence out of the four, following the evolution of the graph we notice that eventually it outperforms all the others after $t = 20$. VNS-greedy2 is once more the slowest of all methods. Neighborhood contributions for neighborhood 2 are unreliable and the majority of improvements seem to be mostly generated from neighborhood 1.

In summary, RVNS-greedy1 appears to consistently converge much faster than the other three methods with significant increases in complexity imposing a lighter burden on it. However we were unable to approach closer than 50% to the competitive values of (El Yafrani and Ahiod 2018) but reliably reached the 100% threshold within a very short time window and in every test. Regardless of the dataset, neighborhood 1 reliably yielded the most contributions.

5.2 Performance Evaluation - Category 2

5.2.1 Performance on the A1 dataset

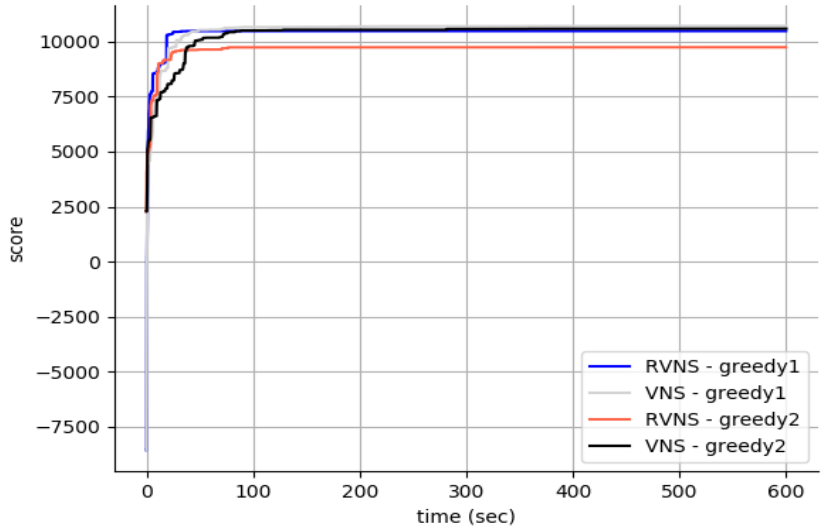

Figure **5.2.9**: Performance of all methods on the A1 dataset.

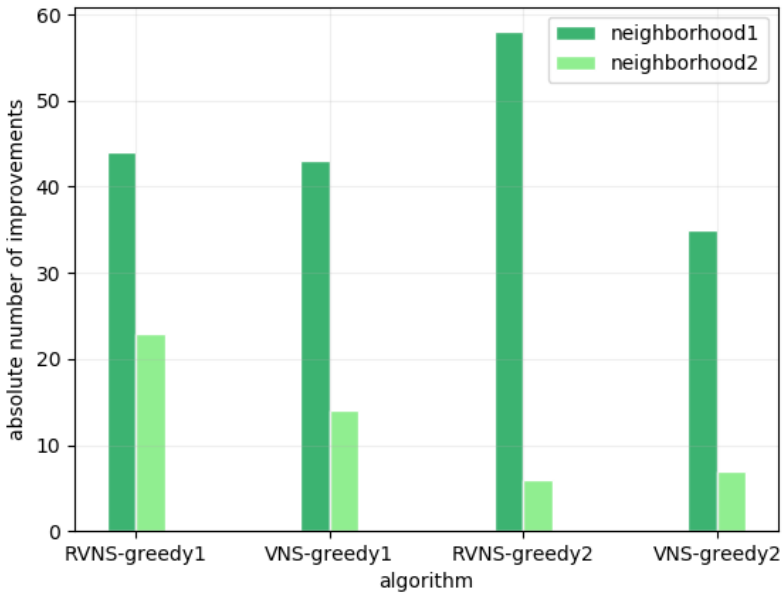

Figure **5.2.10**: Improvements per neighborhood yielded from all methods on the A1 dataset.

method	initial	final (mean)	std. dev.	rsd
RVNS-greedy1	-8560	10524	73	144.16
VNS-greedy1	-8560	10518	76	138.39
RVNS-greedy2	2282	9841	383	25.69
VNS-greedy2	2282	10297	252	40.86

Table **5.3**: Results of the four methods on the A1 dataset.

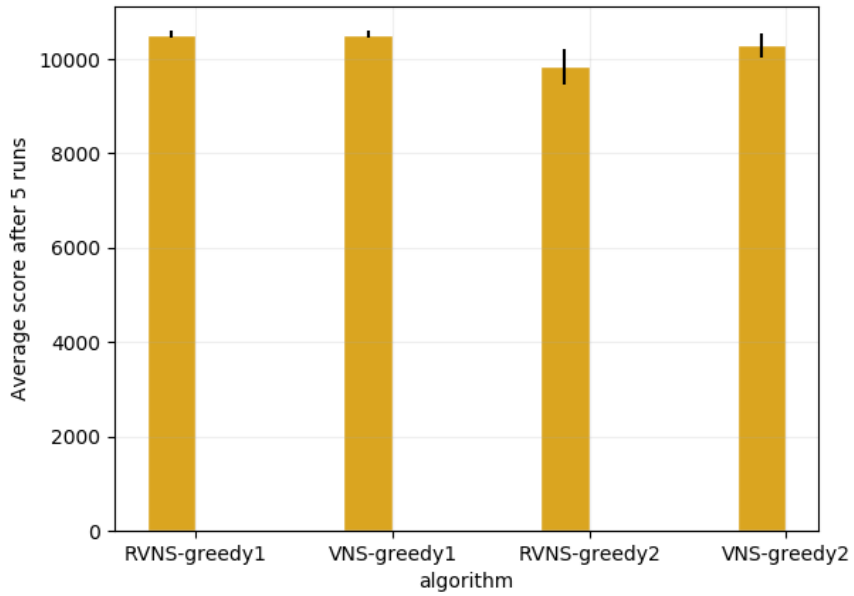

Figure **5.2.11**: Average score of the algorithms and standard deviation after 5 iterations on the A1 dataset.

Table **5.3** and Figure **5.2.11** demonstrate that the best performance on the A1 dataset is achieved with the RVNS-greedy1 algorithm. However, the superiority of RVNS-greedy1 against the two VNS algorithms is only marginal at best. By considering the standard deviation of the four algorithms, we can observe that starting with the greedy1 construction heuristic results in lower standard deviation. That is, algorithms starting with a solution obtained by the greedy2 are less consistent in terms of the final outcome.

We begin by evaluating the performance of the four hybridizations formed on the A1 dataset, which can be perceived to be the simplest of them. From Figure **5.2.9**, it becomes obvious that the greedy2 heuristic (custom) obtains better quality solutions than the greedy1 (grey and blue lines coincide), scoring about 2500 and -8500 respectively.

However, it is worth pointing out that both VNS and RVNS manage to improve the greedy 1 solution to a similar extend, finally reaching the same score values as if they had started with the greedy 2 solution. Moreover, the overall difference in the performance of VNS and RVNS is negligible.

Concerning the time interval needed for the algorithms to converge, we observe that the RVNS gets more easily trapped into local optima compared to the VNS. The time to convergence is about 20 seconds for the RVNS algorithms, while it is two to four times as much for the respective VNS algorithms.

Figure **5.2.10** indicates that both neighborhood structures contribute towards the improvement procedure. However, we can not omit to point out the fact that in all algorithms, the first neighborhood structure has proven to be more helpful. Moreover, algorithms initiated with the greedy2 heuristic benefited less from the second neighborhood compared to their respective algorithms initiated with the solution that was yielded from greedy1. This, though, is to be expected, since the greedy2 heuristic already assimilates the information about the risk of taking a heavy item from a city in the beginning of the route.

5.2.2 Performance on the B1 dataset

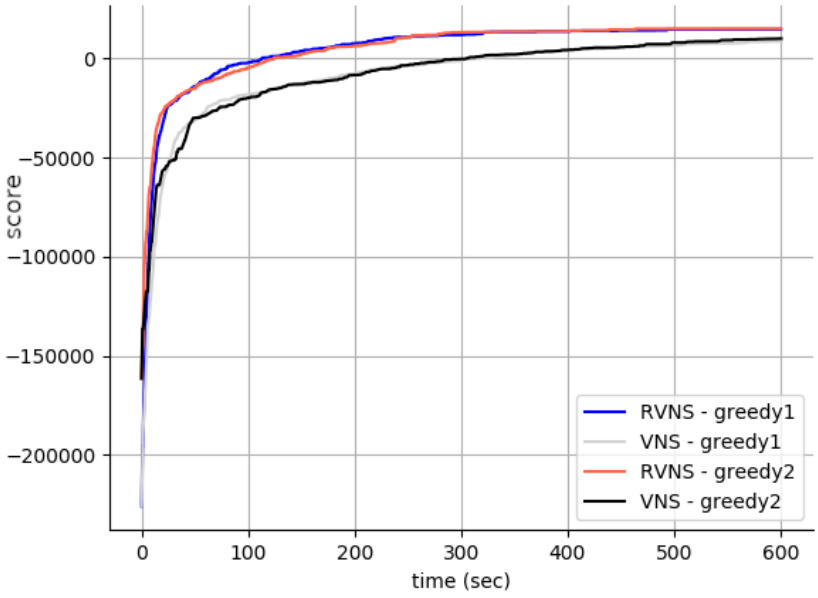

Figure **5.2.12**: Performance of all methods on the B1 dataset.

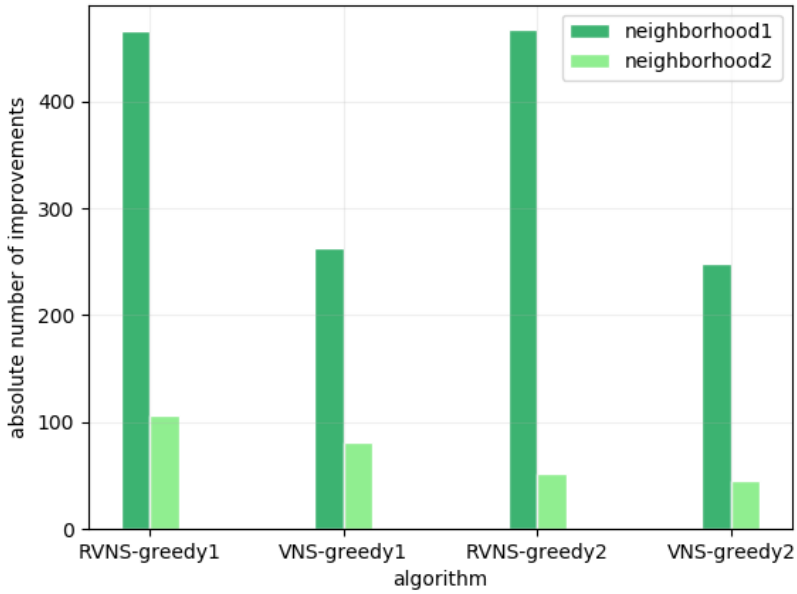

Figure **5.2.13**: Improvements per neighborhood yielded from all methods on the B1 dataset.

method	initial	final (mean)	std. dev.	rsd
RVNS-greedy1	-225758	13735	384	35.77
VNS-greedy1	-225758	4609	996	4.63
RVNS-greedy2	-161349	13779	210	65.61
VNS-greedy2	-161349	1815	2343	0.77

Table **5.4**: Results of the four methods on the B1 dataset.

Table **5.4**, combined with Figure **5.2.14**, indicate that the search driven by RVNS algorithms is more fruitful than the one driven by VNS algorithms. Besides the supremacy of both of them compared to their respective algorithms in terms of the quality of the final solution, which is plainly illustrated, the high standard deviation of VNS methods constitutes yet another disadvantage that RVNS algorithms seem to avoid. Moreover, it should be noticed that the standard deviation of the VNS that incorporated the greedy2 heuristic is far worse than the respective VNS combined with the greedy1.

On the B1 dataset, which comprises 280 cities and 1395 items, i.e., 5 items per city, the performance of the two construction heuristics is similar to that of the A1 dataset. Again, greedy2 yields a better initial solution than greedy1 does. Nevertheless, at the

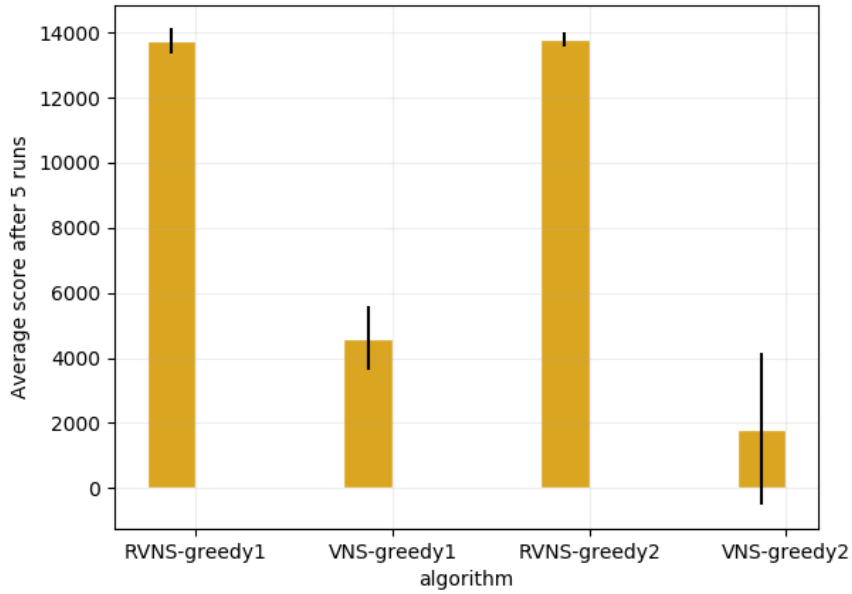

Figure **5.2.14**: Average score of the algorithms and standard deviation after 5 iterations on the B1 dataset.

end of the experiment (after 600 seconds), both VNS and RVNS that started with the greedy2 solution seem to obtain only marginally better final solutions than their respective algorithms that started with greedy1.

Concerning the behavior of VNS against RVNS algorithms, in Figure **5.2.12**, it becomes apparent that RVNS achieves better results during the 600 second time interval. However, VNS seems to demonstrate an aptness to further improve the incumbent solution when both RVNSs seem to be trapped.

Regarding the improvements yielded per neighborhood, again, both of them are proven valuable for all algorithms, with an obvious imbalance that favors the first neighborhood. Nevertheless, in Figure **5.2.13**, a more meaningful insight that can be drawn is the fact that both VNSs, generally, achieve less improvement than RVNSs do. However, this translates only into a marginal difference with respect to the final solution.

5.2.3 Performance on the C1 dataset

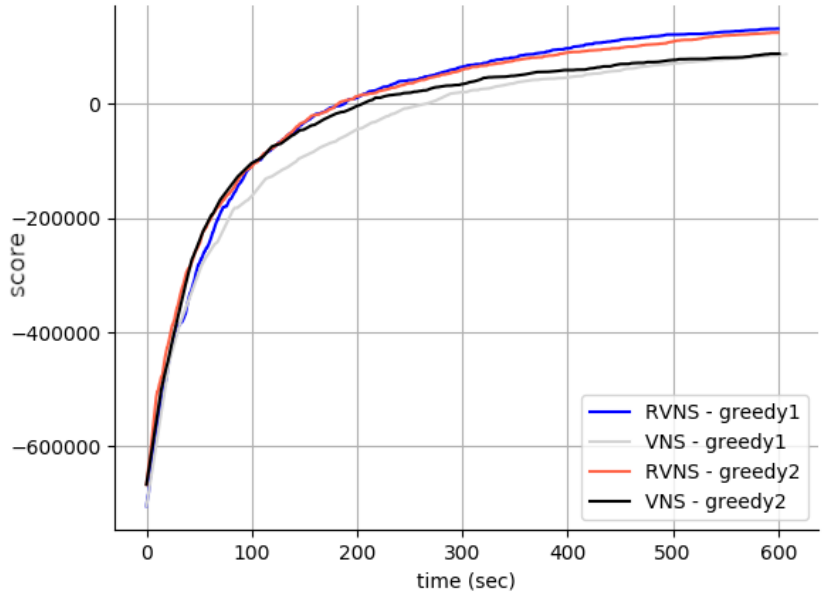

Figure **5.2.15**: Performance of all methods on the C1 dataset.

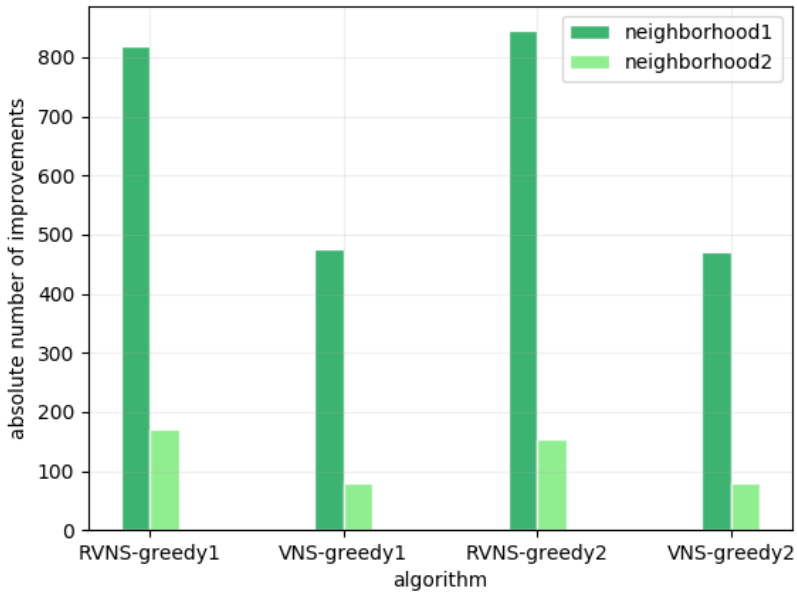

Figure **5.2.16**: Improvements per neighborhood yielded from all methods on the C1 dataset.

method	initial	final (mean)	std. dev.	rsd
RVNS-greedy1	-704805	81086	13065	6.21
VNS-greedy1	-704805	40744	6081	6.70
RVNS-greedy2	-666935	83824	11655	7.19
VNS-greedy2	-666935	60397	8780	6.88

Table **5.5**: Results of the four methods on the C1 dataset.

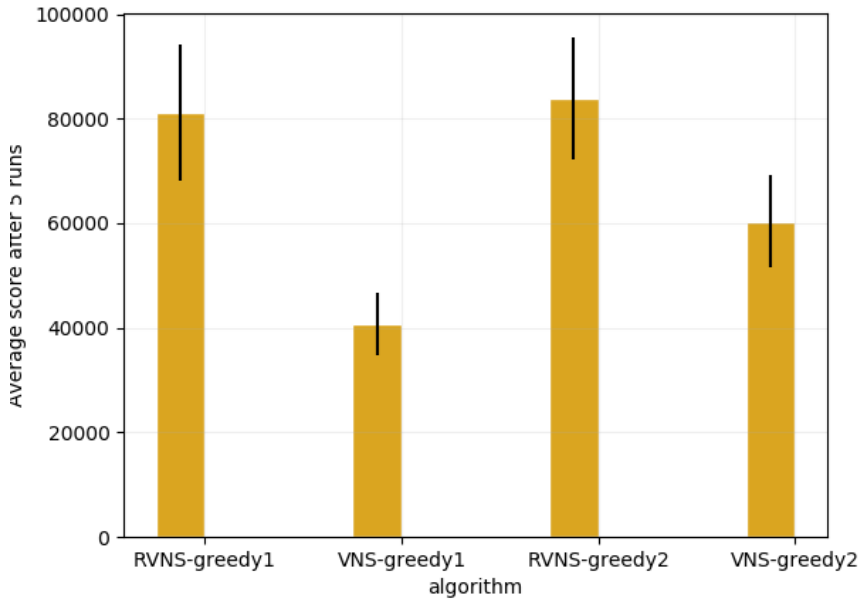

Figure **5.2.17**: Average score of the algorithms and standard deviation after 5 iterations on the C1 dataset.

The insights drawn from Table **5.5** and Figure **5.2.17** demonstrate, once more, that the RVNS algorithms perform generally better on the C1 dataset and, by extension, in all datasets comprising of 280 cities. In this instance, however, we can observe that RVNS algorithms exhibit higher standard deviation compared to what we derive in previous analyses. Within the 600 second interval, the improvement on the initial solution is substantial under all four algorithms. In our opinion, this fact is certainly noteworthy.

The C1 dataset contains 280 cities and 2790 items, i.e., 10 items per city. Consequently, the computational load the solvers have to process is critically increased. This is the reason why no obvious convergence of the four algorithms in Figure **5.2.15** is observed. What is interesting to point out though, is once again the superiority of the greedy2 heuristic against greedy1. The same applies for the RVNSs over VNSs.

Another insight that can be drawn from the same figure is the extraordinary improvement of the initial solution, with the latter scoring -750,000 (greedy1) and, after the application of the algorithms, scoring 150,000 (RVNS-greedy1).

Figure **5.2.16** demonstrates the efficient utilization of the two neighborhood structures. In line with the analysis on the B2 dataset, RVNSs yield more improvements. However, in this instance of the problem, an analogous behavior of the RVNSs is clearly observed, with improvements obtained from the first and second neighborhoods being almost the same. This is also true for VNSs.

5.2.4 Performance on the A2 dataset

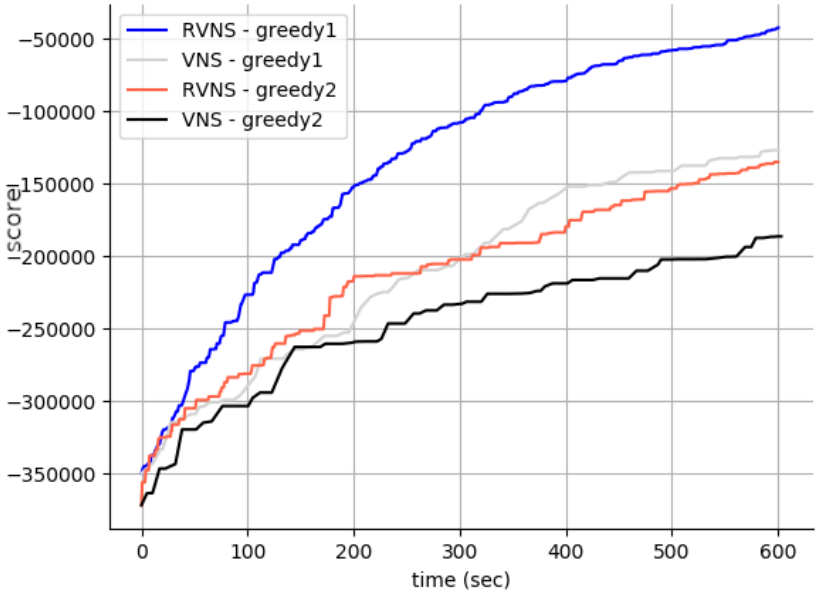

Figure **5.2.18**: Performance of all methods on the A2 dataset.

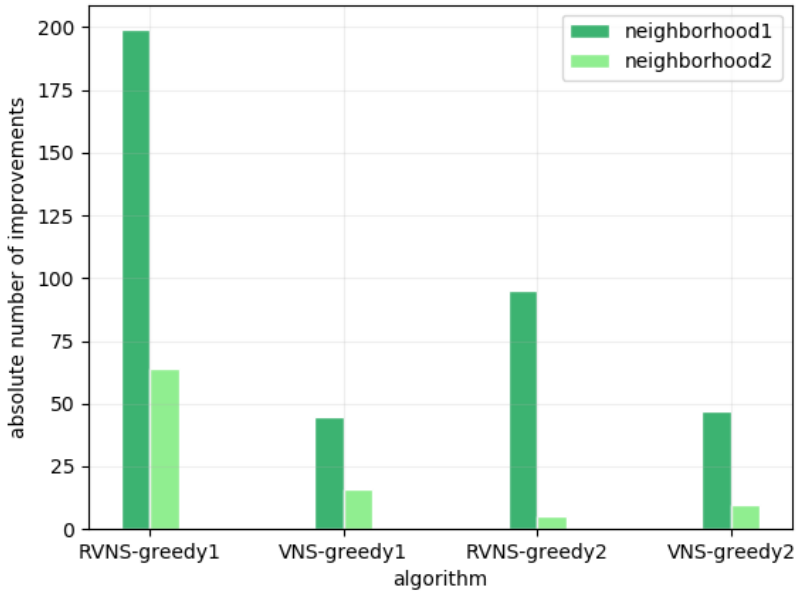

Figure **5.2.19**: Improvements per neighborhood yielded from all methods on the A1 dataset.

method	initial	final (mean)	std. dev.	rsd
RVNS-greedy1	-349964	-48182	4897	9.84
VNS-greedy1	-349964	-146246	20296	7.21
RVNS-greedy2	-371981	-146173	12589	11.61
VNS-greedy2	-371981	-168414	6895	24.43

Table **5.6**: Results of the four methods on the A2 dataset.

A2 is the first dataset where the greedy1 heuristic obtains a better solution than the greedy2 algorithm, as depicted in Figure **5.2.18**. In fact, this directly translates into a superiority of the RVNS-greedy1 over the rest of the algorithms. However, since the A2 dataset is comprised of 4461 cities, the time interval of 600 seconds appears to be too short for any algorithmic convergence to become apparent. Such problem instances would strongly benefit from parallelization techniques which are, however, are outside the scope of this thesis.

Concerning the improvements per neighborhood, neighborhood 1 is, apparently, the one that most frequently yields improvements to the incumbent solution. We can not ignore, though, the difference in total improvements being achieved with RVNS-greedy1

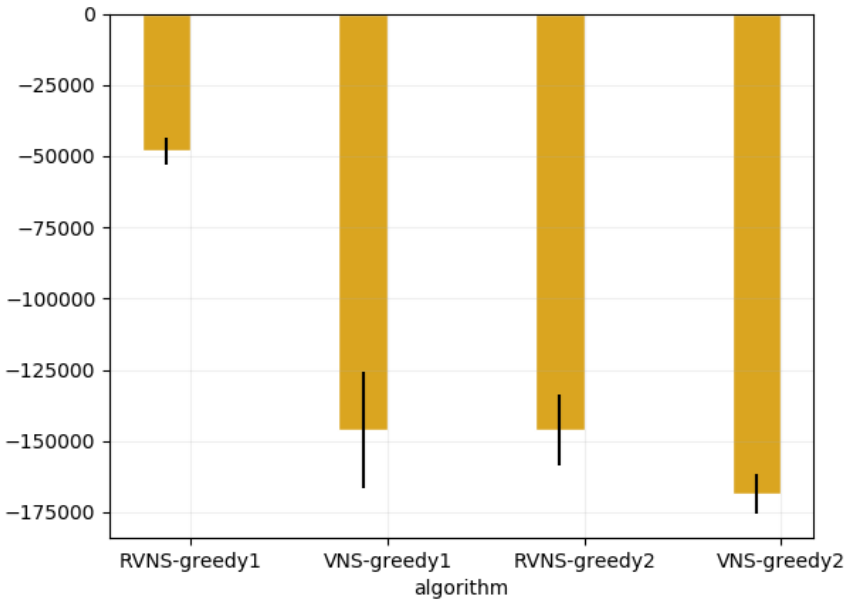

Figure **5.2.20**: Average score of the algorithms and standard deviation after 5 iterations on the A2 dataset.

and the rest of the methodologies; despite the latter managing dramatically fewer than the first, as Figure **5.2.19** clearly illustrates.

5.2.5 Performance on the B2 dataset

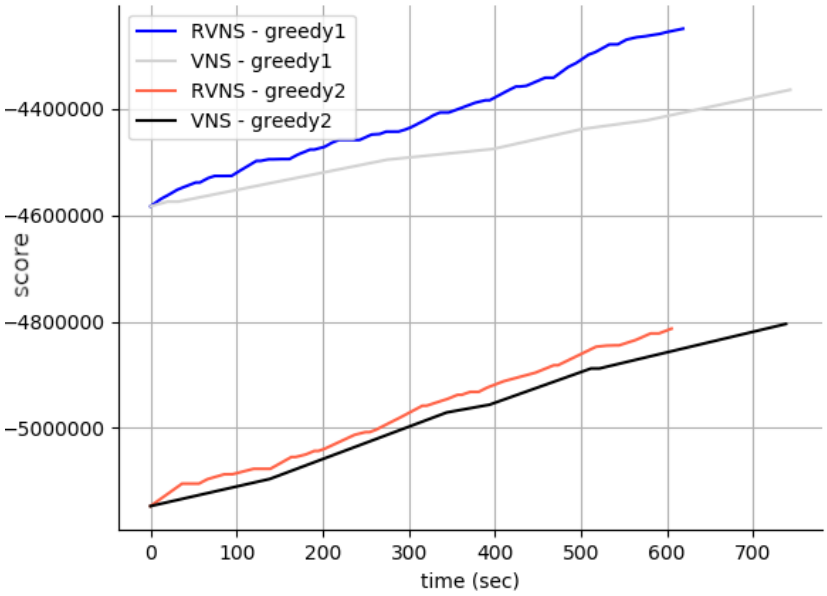

Figure **5.2.21**: Performance of all methods on the B2 dataset.

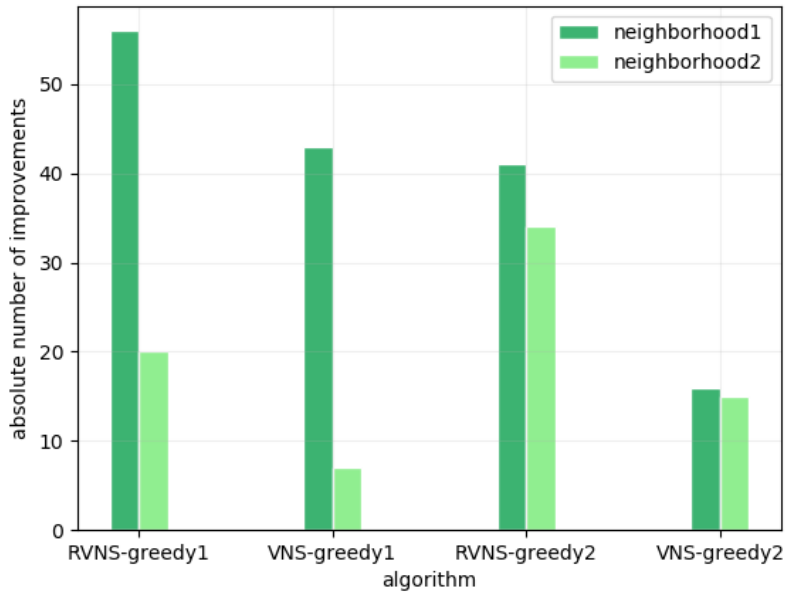

Figure **5.2.22**: Improvements per neighborhood yielded from all methods on the B2 dataset.

method	initial	final (mean)	std. dev.	rsd
RVNS-greedy1	-4583529	-4266576	5451	782.71
VNS-greedy1	-4583529	-4313695	21888	197.08
RVNS-greedy2	-5146634	-4848423	33557	144.48
VNS-greedy2	-5146634	-4868431	28857	168.71

Table **5.7**: Results of the four methods on the B2 dataset.

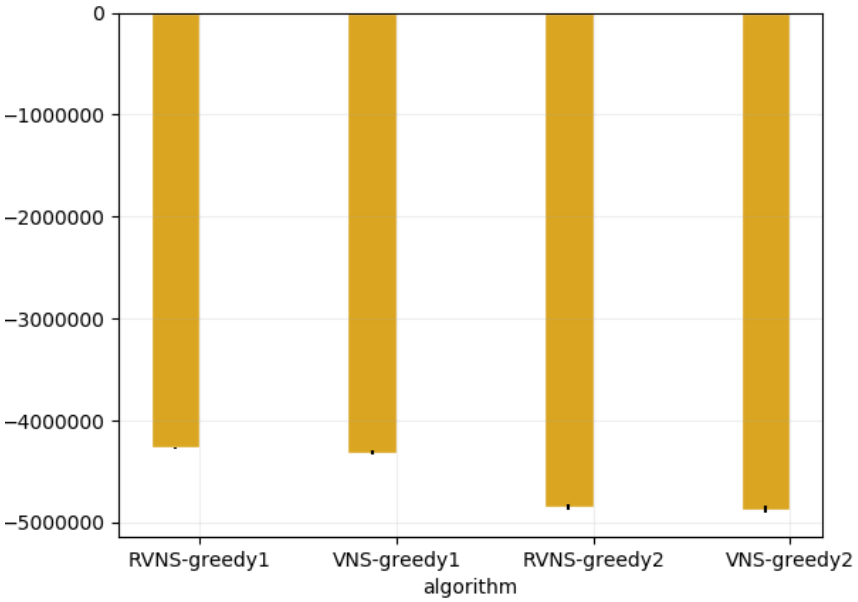

Figure **5.2.23**: Average score of the algorithms and standard deviation after 5 iterations on the B2 dataset.

Yet again, according to Figure **5.2.21**, the greedy1 construction heuristic yields better solutions than greedy2 and, more specifically, this is the biggest difference in solution quality between them that has been observed this far. Out of all four algorithms, RVNS-greedy1 performs substantially better, while both RVNSs obtain a higher quality solution within the fixed 600 second time interval than the respective VNS counterparts that started from the same point.

Regarding the distribution of improvements per neighborhood structure, it is clearly demonstrated that the first neighborhood yields more improvements during the search phase, with the second neighborhood contributing less. This, however, does not indicate that the second neighborhood is less useful that the first. On the contrary, in VNS algorithms, neighborhood structures act collaboratively and, should either of those be

omitted, the final outcome will be considerably worse.

5.2.6 Performance on the C2 dataset

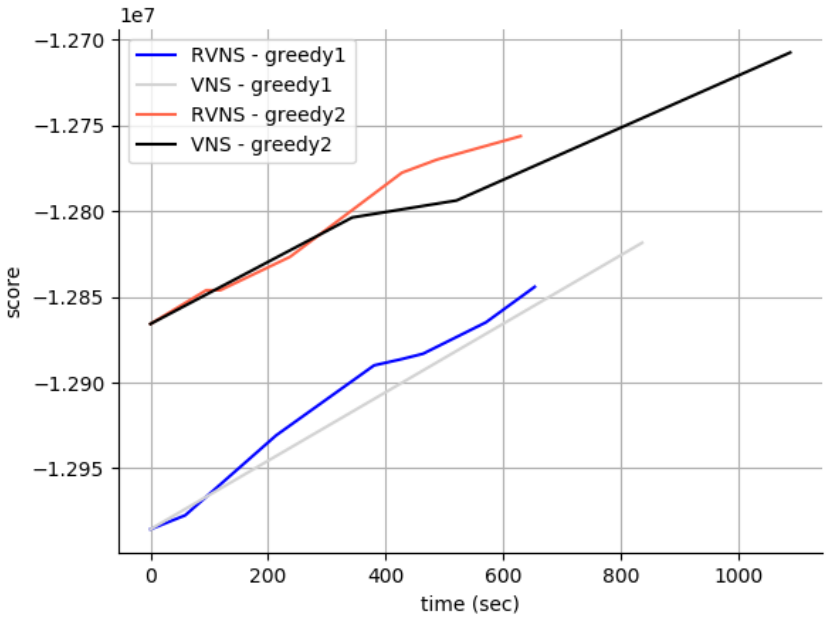

Figure **5.2.24**: Performance of all methods on the C2 dataset.

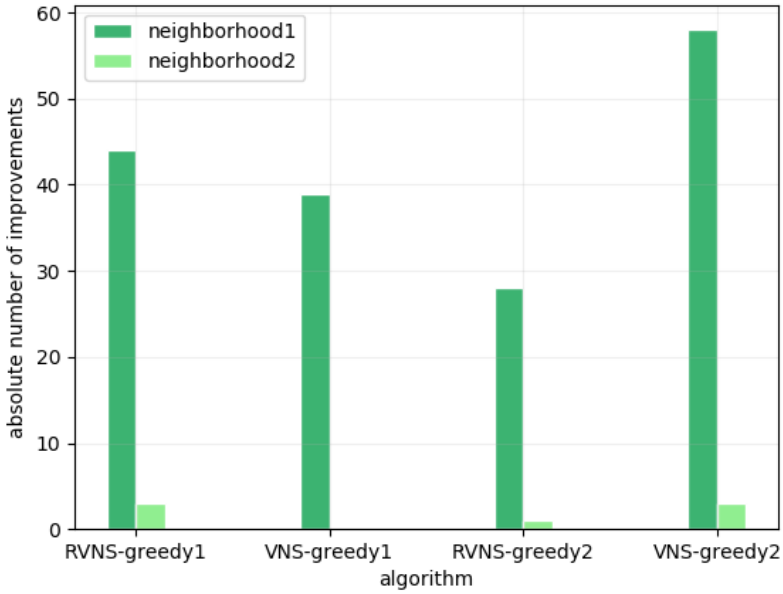

Figure **5.2.25**: Improvements per neighborhood yielded from all methods on the C2 dataset.

method	initial	final (mean)	std. dev.	rsd
RVNS-greedy1	-12985558	-12838346	41604	308.58
VNS-greedy1	-12985558	-12594274	168483	74.75
RVNS-greedy2	-12865814	-12716186	8034	1582.80
VNS-greedy2	-12865814	-12676728	36301	349.21

Table **5.8**: Results of the four methods on the C2 dataset.

According to Figure **5.2.24**, the greedy2 heuristic obtains a significantly better solution than greedy1 does. More importantly, both algorithms that initiated from the greedy2 solution seem to outperform their greedy1 counterparts. Additionally, within the 600 second time interval, RVNSs yield higher quality solutions than VNSs do. However, it should be noted that the selected time interval is, once again, too short to tackle such an enormous problem instance. Parallelization techniques will be significantly useful in overcoming the limitations imposed if the time interval is required to remain within the boundaries of such a short span.

As per Figure **5.2.25**, the improvements on the incumbent solution resulting from the two neighborhoods are vastly favoring the first. However, as can be seen in the same

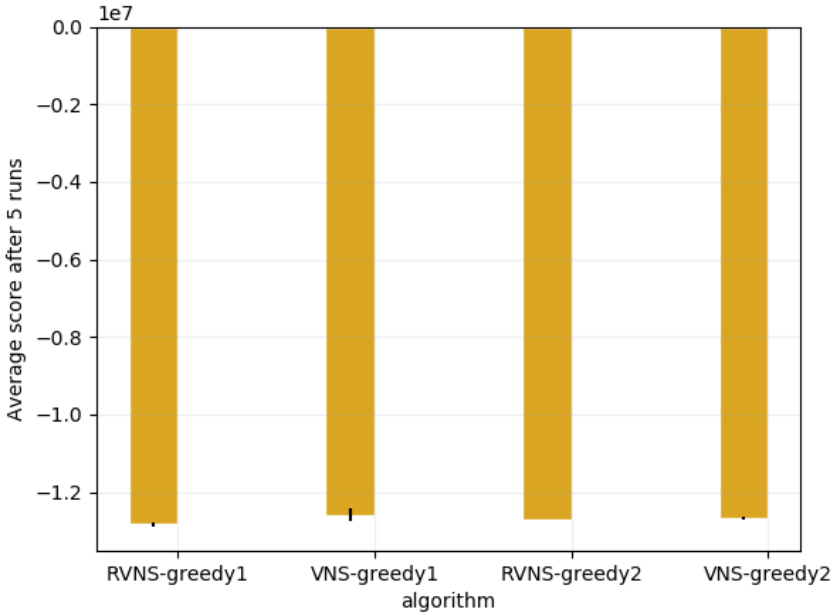

Figure **5.2.26**: Average score of the algorithms and standard deviation after 5 iterations on the C2 dataset.

Figure, the absolute number of improvements under all four approaches is quite low.

If we were to select only one among the four algorithms, the RVNS-greedy1 would probably be our choice. That is, in addition to its good performance on the smaller datasets with 280 cities, RVNS-greedy1 also outperforms the rest of the algorithms in the bigger problem instances. In terms of standard deviation, once again RVNS-greedy1 proves to outperform the other algorithms, except for the final and larger dataset, i.e., the C2 dataset. It is quite notable that in the same dataset the VNS-greedy1 manages to improve the initial solution by almost 400,000 on average. However, according to the values that were acquired from our experimentation, the same algorithm consistently yields the highest standard deviation, quadruple that of the algorithm with the second highest.

CHAPTER 6

Statistical Analysis

This chapter intends to analyze the behavior of our algorithmic implementations from a statistical standpoint. In particular, we focus the analysis on the four datasets pertaining to Category 1; namely, the *eli76*, *kroA100*, *ch130* and *u159* datasets. We intentionally omit a thorough examination of our the performance of our algorithm on the rest of the datasets of Category 2 since, as already cited in Chapter 5, competitive algorithms for the latter category utilize sophisticated techniques for acceleration and to reduce the solution space, an aspect that extends beyond the scope of this thesis.

Performance-wise, approximate algorithms are used to analyze their distances from the optimal values (if there exist), or from a decent known value. This distance is usually expressed as in terms of percentage and in our particular case this distance constitutes the variable under examination.

For statistical validity reasons, we run all four of our algorithms for a total of 50 iterations on each dataset. The runtime was determined by convergence indications drawn from Figures **5.1.1**, **5.1.3**, **5.1.5** and **5.1.7** in Chapter 5. The approach we follow is (1) to present some descriptive statistical measures on the datasets produced, (2) to graphically represent the distribution of errors (i.e., distance) from a competitive value and (3) to conduct a normality test that informs us whether the samples follow the normal distribution or not. The coefficient of significance α that we used during this test was set to 0.05.

6.1 Analysis of errors on the eli76 dataset

According to Table **6.1**, distances from the competitive value are similar among all algorithms. In particular, the minimum values are almost identical, while slight alter-

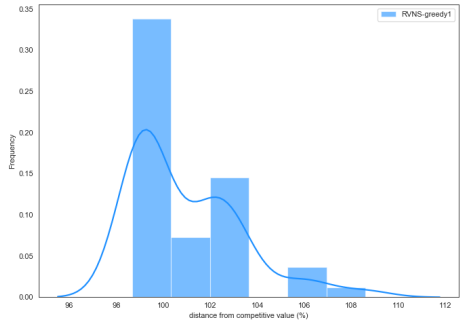

Figure **6.1.1**: Offset distribution - RVNS-greedy1.

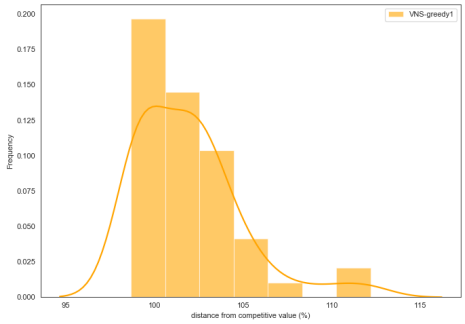

Figure **6.1.2**: Offset distribution - VNS-greedy1.

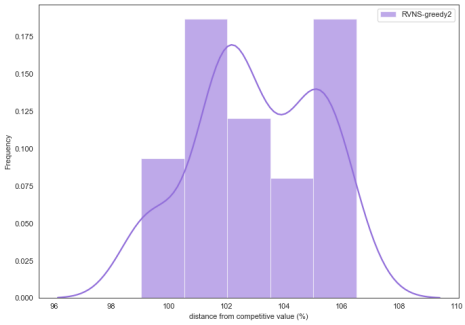

Figure **6.1.3**: Offset distribution - RVNS-greedy2.

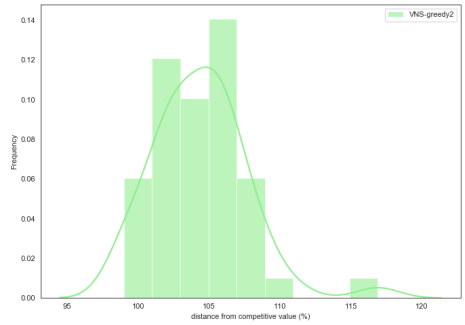

Figure **6.1.4**: Offset distribution - VNS-greedy2.

method	min	max	mean	variance	skewness	kurtosis
RVNS-greedy1	99.45	103.57	100.40	0.93	1.23	1.18
VNS-greedy1	99.45	105.06	100.79	1.51	1.41	2.35
RVNS-greedy2	99.60	102.71	101.26	0.78	-0.20	-0.91
VNS-greedy2	99.58	107.02	101.83	1.90	0.95	2.46

Table **6.1**: Descriptive statistics of the errors for the *eli76* dataset.

ations are noticeable for the maximum values. Moreover, variance seems to be quite low in all cases. Skewness and Kurtosis metrics are used to further enhance the verdict of the statistical test for normality, presented in Table **6.2**. We can observe that only the RVNS-greedy2 heuristic passes this test, i.e., we cannot reject the null hypothesis that the sample follows a normal distribution. However, such verdicts cannot be taken for

method	Statistic value	p-value	null hypothesis
RVNS-greedy1	14.37	7×10^{-4}	rejected
VNS-greedy1	20.38	3.75×10^{-5}	rejected
RVNS-greedy2	3.99	0.14	not rejected
VNS-greedy2	14.55	7×10^{-4}	rejected

Table **6.2**: Normality test of errors for the *eli76* dataset.

granted, since Figure **6.1.3** is not aligned with this outcome. In general, Figures **6.1.1** to **6.1.4** demonstrate that errors follow distributions which, in most cases, demonstrate a right tail. This indicates the existence of outlier values that represent experiments where algorithms performed worse than the average.

6.2 Analysis of errors on the kroA100 dataset

method	min	max	mean	variance	skewness	kurtosis
RVNS-greedy1	95.30	100.59	97.06	1.34	0.86	0.91
VNS-greedy1	95.56	109.25	98.65	5.44	2.20	7.06
RVNS-greedy2	95.30	101.78	98.16	2.55	0.01	-0.65
VNS-greedy2	96.38	108.67	99.75	5.66	1.52	2.76

Table **6.3**: Descriptive statistics of the errors for the *kroA100* dataset.

method	Statistic value	p-value	null hypothesis
RVNS-greedy1	8.73	0.01	rejected
VNS-greedy1	41.32	1.06×10^{-9}	rejected
RVNS-greedy2	1.02	0.6	not rejected
VNS-greedy2	23	1.01×10^{-5}	rejected

Table **6.4**: Normality test of errors for the *kroA100* dataset.

In the *kroA100* dataset the four algorithms, again, perform similarly in terms of mean values. However, a careful examination of Table **6.3** reveals that the two VNS algorithms have a considerably higher variance compared to the RVNS algorithms. This might indicate that VNS takes a little longer than RVNS to converge. The fact that errors produced by RVNS-greedy2 seem once again to follow a normal distribution, as per Table **6.4**, is quite interesting. This time, however, this fact is also supported by Figure **6.2.7**.

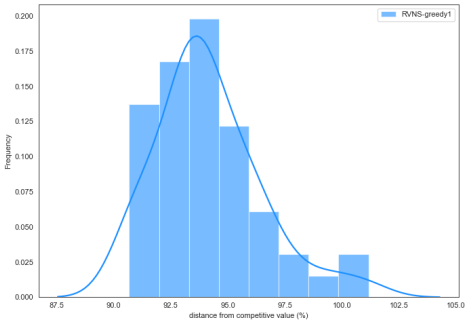

Figure **6.2.5**: Offset distribution - RVNS-greedy1.

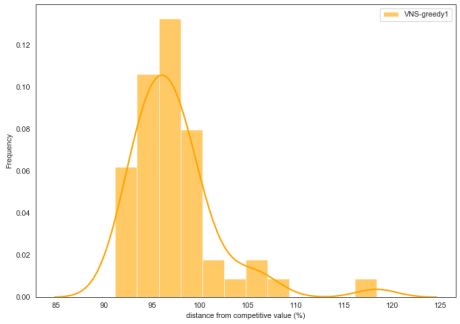

Figure **6.2.6**: Offset distribution - VNS-greedy1.

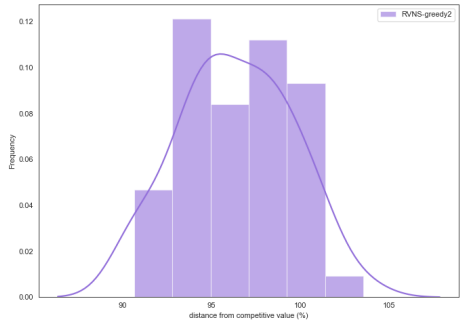

Figure **6.2.7**: Offset distribution - RVNS-greedy2.

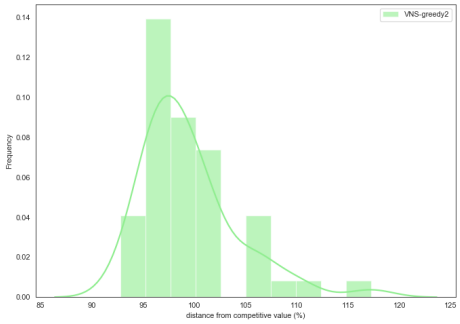

Figure **6.2.8**: Offset distribution - VNS-greedy2.

The rest of the error distributions (see Figures **6.2.5**, **6.2.6**, **6.2.8**) demonstrate similar results as previously, i.e., in the *eli76* dataset.

6.3 Analysis of errors on the ch130 dataset

method	min	max	mean	variance	skewness	kurtosis
RVNS-greedy1	49.61	59.04	53.51	4.52	0.78	0.49
VNS-greedy1	50.56	72.01	55.84	20.58	1.77	3.02
RVNS-greedy2	49.65	99.12	54.42	46.57	5.81	35.41
VNS-greedy2	52.45	103.29	61.81	115.51	2.11	4.10

Table **6.5**: Descriptive statistics of the errors for the *ch130* dataset.

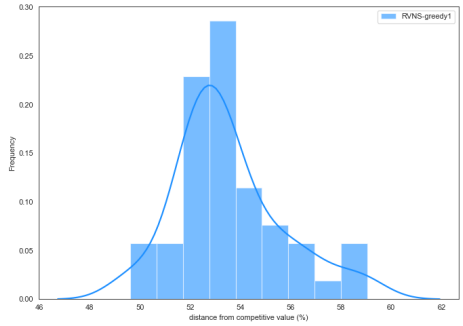

Figure **6.3.9**: Offset distribution - RVNS-greedy1.

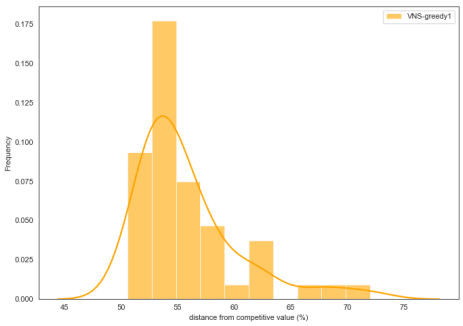

Figure **6.3.10**: Offset distribution - VNS-greedy1.

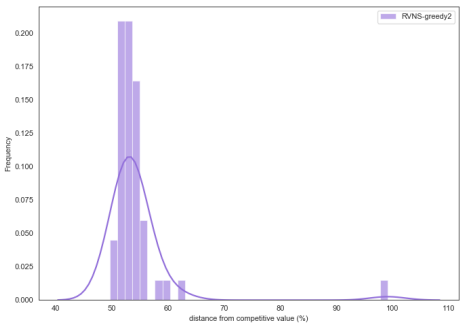

Figure **6.3.11**: Offset distribution - RVNS-greedy2.

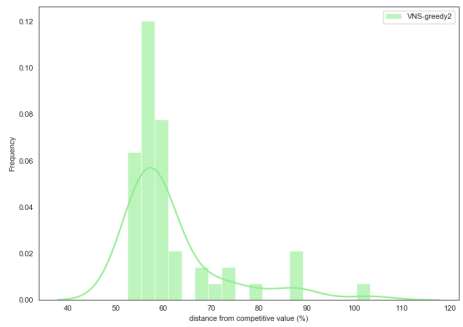

Figure **6.3.12**: Offset distribution - VNS-greedy2.

method	Statistic value	p-value	null hypothesis
RVNS-greedy1	6.59	0.03	rejected
VNS-greedy1	27.17	1.25×10^{-6}	rejected
RVNS-greedy2	100.58	1.44×10^{-22}	rejected
VNS-greedy2	34.26	3.62×10^{-8}	rejected

Table **6.6**: Normality test of errors for the *ch130* dataset.

The performance of our algorithms in the *ch130* dataset is most promising, except from the large variance values depicted in Table **6.5**. In the same table, it can be easily observed that the mean errors are close to 50% in most cases. This is a quite encouraging result, especially considering that our algorithms run for just a few seconds to produce these outcomes. The large fluctuations, though, prevent the error populations from fol-

lowing a normal distribution, as indicated in both Table **6.6** and Figures **6.3.9** to **6.3.12**. Finally, it is worth noting that algorithms that have started with the greedy2 solution yield a substantially higher variance. That indicates, either these algorithms required a bigger time-interval to converge, their performance is less stable or that the latter is a product of the former.

6.4 Analysis of errors on the u159 dataset

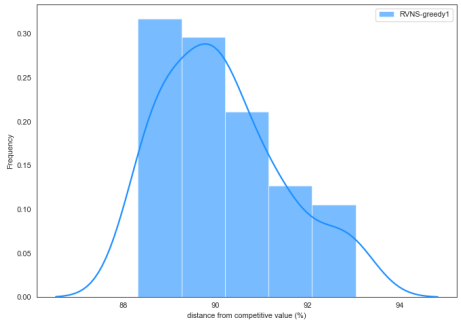

Figure **6.4.13**: Offset distribution - RVNS-greedy1.

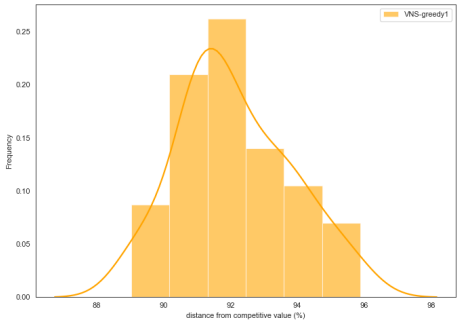

Figure **6.4.14**: Offset distribution - VNS-greedy1.

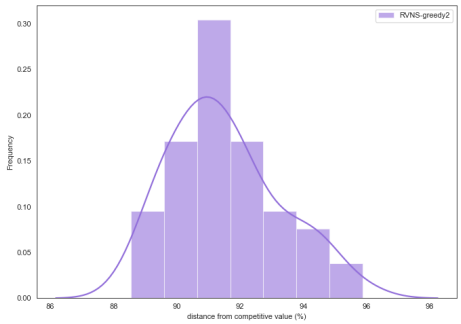

Figure **6.4.15**: Offset distribution - RVNS-greedy2.

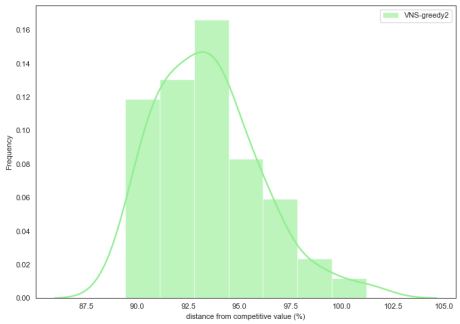

Figure **6.4.16**: Offset distribution - VNS-greedy2.

The performance of our algorithms is also encouraging on the *u159* dataset. Except from the VNS-greedy2 metaheuristic, all of our methods yield errors that follow the normal distribution, as Table **6.8** demonstrates. That can be interpreted as, the time interval selected (30 seconds) being adequate to achieve algorithmic convergence, while competitive values in the literature are obtained after much longer intervals. As per

method	min	max	mean	variance	skewness	kurtosis
RVNS-greedy1	88.31	93.04	90.13	1.72	0.64	-0.39
VNS-greedy1	89.02	95.89	92.12	2.82	0.33	-0.48
RVNS-greedy2	88.54	95.89	91.51	3.04	0.55	-0.39
VNS-greedy2	89.41	101.20	93.43	6.38	0.78	0.57

Table **6.7**: Descriptive statistics of the errors for the *u159* dataset.

method	Statistic value	p-value	null hypothesis
RVNS-greedy1	3.93	0.14	not rejected
VNS-greedy1	1.40	0.49	not rejected
RVNS-greedy2	2.94	0.23	not rejected
VNS-greedy2	6.81	0.03	rejected

Table **6.8**: Normality test of errors for the *u159* dataset.

Figures **6.4.13** to **6.4.16**, we can observe the phenomenon of significant presence of outliers being absent, further supporting the statements above.

CHAPTER 7

Conclusions

Drawing from the statistical analysis in Chapter 6 it appears that both VNS and RVNS can be successfully applied to the Travelling Thief Problem, the latter performing slightly better in most, including large, problem instances and frequently passing the normality test in smaller datasets. Thus, the instance of the problem, and more specifically the size of the instance, seem to be the major limiting factors when unmodified RVNS is applied under the scope of strict time limitations. The starting conditions also appear to play a significant role in the quality of the outcome, since in our tests the different initial heuristic solutions, which are generated as in input, seem to narrow down the search space exploration capabilities.

7.1 Summary

In this thesis we provide a preliminary basis to demonstrate the concept of applying RVNS to the Travelling Thief Problem. While the obtained results are lacking in comparison to those obtained in relevant literature for the same datasets with more advanced performance enhancement methods, our approach is significantly simpler as it only requires the implementation of a construction heuristic and RVNS while achieving decent results in the span of the same timeframes. Furthermore we observe that RVNS, especially when combined with a construction heuristic like greedy1, is, in our particular case, very efficient in achieving results that, while not up to competitive solution standards, can be obtained remarkably fast. This demonstrates a potential for RVNS in areas where frequent and rapid re-calculation of efficient solutions is more important that the actual target value of the objective function.

7.2 Study Limits - Constraints

In the interest of objectivity and reproducibility of the results, some of the more technical aspects concerning the code implementation for the obtainment of the results need to be presented.

All experimentation was done with Python 3.7.3 which at the time was the most recent stable version of the 3.x series available. No attempt for Parallelization has been undertaken, although because of the prevalence of loop control flow structures in the implementation of RVNS, the author believes it could greatly benefit the processing speed in multicore and multi-socket systems.

7.3 Future Work

As the resulting figures in *Chapter 5* demonstrate, the quality of the initial solution obtained by the construction function has a decisive effect of the outcome of the main heuristic. We therefore consider that further research on the construction function heuristics to establish a "good" starting point possesses significant potential to greatly improve the quality and outcomes in the Travelling Thief Problem. Drawing inspiration from the limitations of the current study, another area which presents great potential for future research is the utilization of parallel processing to increase the total processing done per unit of time. Considering that RVNS iterates through many loop flow structures there is significant potential for parallelization. While this will not demonstrate a significant effect in loop-limited repetition scenarios, it has great potential in improving the outcomes in time-based scenarios by reaching a better solution in the same units of time. Lastly, a combination of the two research directions identified above also presents a promising area for significant contributions in future research with potential industry applications. The combination of more advanced construction functions, possibly tailored to the problem, with parallelization in the main solver heuristic could yield significantly improved solutions in a fraction of the time, translating to appreciable improvement in efficiency in scenarios where the variables of the problem are dynamic and new solutions need to be retroactively calculated.

Bibliography

Abdmouleh, Z., A. Gastli, L. Ben-Brahim, M. Haouari, and N. A. Al-Emadi (2017). Review of optimization techniques applied for the integration of distributed generation from renewable energy sources. *Renewable Energy 113*, 266–280.

Applegate, D., W. Cook, and A. Rohe (2003). Chained Lin-Kernighan for Large Traveling Salesman Problems. *INFORMS Journal on Computing 15*(1), 82–92.

Bonyadi, M. R., Z. Michalewicz, and L. Barone (2013). The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In *2013 IEEE Congress on Evolutionary Computation*, pp. 1037–1044. IEEE.

Bonyadi, M. R., Z. Michalewicz, M. R. Przybylek, and A. Wierzbicki (2014). Socially Inspired Algorithms for the Travelling Thief Problem. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pp. 421–428. ACM.

Burke, E. K. and G. Kendall (2013). *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques* (2nd ed.). Springer Publishing Company, Incorporated.

Cafieri, S., P. Hansen, and N. Mladenović (2014). Edge-ratio network clustering by Variable Neighborhood Search. *The European Physical Journal B 87*(5), 116.

Cordeau, J.-F., M. Gendreau, A. Hertz, G. Laporte, J.-F. Cordeau, and J.-S. Sormany (2004). Les Cahiers du GERAD ISSN: 0711 - 2440 New Heuristics for the Vehicle Routing Problem.

Deb, K. (2014). Multi-Objective Optimization. In *Search methodologies*, pp. 403–449. Springer.

Dorigo, M. and G. D. Caro (1999). Ant Colony Optimization: A New Meta-Heuristic. *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406) 2*, 1470–1477 Vol. 2.

Duarte, A., J. Sánchez-Oro, N. Mladenović, and R. Todosijević (2018). Variable Neighborhood Descent. *Handbook of Heuristics*, 341–367.

El Yafrani, M. and B. Ahiod (2016). Population-based vs. Single-solution Heuristics for the Travelling Thief Problem. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pp. 317–324. ACM.

El Yafrani, M. and B. Ahiod (2017). A local search based approach for solving the travelling thief problem. *Applied Soft Computing 52*, 795–804.

El Yafrani, M. and B. Ahiod (2018). Efficiently solving the traveling thief problem using hill climbing and simulated annealing. *Information Sciences 432*, 231–244.

Faulkner, H., S. Polyakovskiy, T. Schultz, and M. Wagner (2015). Approximate Approaches to the Traveling Thief Problem. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 385–392. ACM.

Ferentinos, K. P., K. G. Arvanitis, and N. Sigrimis (2002). Heuristic optimization methods for motion planning of autonomous agricultural vehicles. *Journal of Global Optimization 23*(2), 155–170.

Goldberg, D. E., R. Lingle, et al. (1985). Alleles, Loci, and the Traveling Salesman Problem. In *Proceedings of an international conference on genetic algorithms and their applications*, Volume 154, pp. 154–159. Lawrence Erlbaum, Hillsdale, NJ.

Haimes, Y. (1971). On a Bicriterion Formulation of the Problems of Integrated System Identification and System Optimization. *IEEE transactions on systems, man, and cybernetics 1*(3), 296–297.

Hansen, P., N. Mladenović, and J. A. M. Pérez (2010). Variable neighbourhood search: Methods and applications. *Annals of Operations Research 175*(1), 367–407.

Hansen, P., N. Mladenović, and D. Perez-Britos (2001). Variable Neighborhood Decomposition Search. *Journal of Heuristics 7*(4), 335–350.

Hoos, H. H. and T. Stützle (2004). *Stochastic Local Search : Foundations and Applications*. Elsevier.

Huang, C.-Y. R., C.-Y. Lai, and K.-T. T. Cheng (2009). Chapter 4 - fundamentals of algorithms. In L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng (Eds.), *Electronic Design Automation*, pp. 173 – 234. Boston: Morgan Kaufmann.

Leeuwen, J. V., A. Meyer, M. Nival, et al. (1990). Handbook of Theoretical Computer Science: Algorithms and Complexity.

Lusa, A. and C. N. Potts (2008). A Variable Neighbourhood Search Algorithm for the Constrained Task Allocation Problem. *Journal of the Operational Research Society 59*(6), 812–822.

Martello, S., D. Pisinger, and P. Toth (1999). Dynamic Programming and Strong Bounds for the 0-1 Knapsack Problem. *Management Science 45*(3), 414–424.

Mei, Y., X. Li, and X. Yao (2014). Improving Efficiency of Heuristics for the Large Scale Traveling Thief Problem. In *Asia-Pacific Conference on Simulated Evolution and Learning*, pp. 631–643. Springer.

Mei, Y., X. Li, and X. Yao (2016). On Investigation of Interdependence Between Subproblems of the Travelling Thief Problem. *Soft Computing 20*(1), 157–172.

Michalewicz, Z. (2012). Quo Vadis, Evolutionary Computation?: On a Growing Gap Between Theory and Practice. In *Proceedings of the 2012 World Congress Conference on Advances in Computational Intelligence*, WCCI'12, Berlin, Heidelberg, pp. 98–121. Springer-Verlag.

Mühlenbein, H. (1991). Evolution in Time and Space The Parallel Genetic Algorithm. In *Foundations of genetic algorithms*, Volume 1, pp. 316–337. Elsevier.

Pérez, J. A. M., N. Mladenović, B. M. Batista, and I. J. G. del Amo (2006). Variable Neighbourhood Search. In *Metaheuristic Procedures for Training Neutral Networks*, pp. 71–86. Springer.

Polyakovskiy, S., M. R. Bonyadi, M. Wagner, Z. Michalewicz, and F. Neumann (2014a). A Comprehensive Benchmark Set and Heuristics for the Traveling Thief Problem. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pp. 477–484. ACM.

Polyakovskiy, S., M. r. Bonyadi, M. Wagner, Z. Michalewicz, and F. Neumann (2014b, 07). Manual for the ieee cec 2014 competition optimisation of problems with multiple interdependent components.

Reinelt, G. (1991). TSPLIB - A Traveling Salesman Problem Library. *ORSA Journal on Computing 3*(4), 376–384.

Resende, M. G. C. and C. C. Ribeiro (2010). Chapter 11 GRASP : Greedy Randomized Adaptive Search Procedures.

Stützle, T., M. Dorigo, et al. (1999). ACO Algorithms for the Traveling Salesman Problem. *Evolutionary algorithms in engineering and computer science*, 163–183.

Wagner, M., M. Lindauer, M. Mısır, S. Nallaperuma, and F. Hutter (2018). A case study of algorithm selection for the traveling thief problem. *Journal of Heuristics 24*(3), 295–320.

Wu, J., M. Wagner, S. Polyakovskiy, and F. Neumann (2017). Exact approaches for the travelling thief problem. In *Asia-Pacific Conference on Simulated Evolution and Learning*, pp. 110–121. Springer.