UNIVERSITY OF PRIMORSKA
FACULTY OF MATHEMATICS, NATURAL SCIENCES AND
INFORMATION TECHNOLOGIES KOPER

Milan Djordjevic

GRAFTED GENETIC ALGORITHM AND THE TRAVELING VISITOR PROBLEM

PhD Thesis

March 2012

Supervisor: Prof. Andrej Brodnik, PhD

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my Director of Studies, Prof. Andrej Brodnik. He has helped, guided and supported me throughout this work. I am deeply indebted to Andrej for his generous and invaluable advice on various matters, be it writing papers, giving talks, preparing slides, or other non-research matters. I am particularly thankful for his patience and encouragement, which has allowed me to pursue what has interested me.

I am also very grateful and extend my sincere thanks to Dr. Dragan Marušič, Dr. Branko Kavšek, Dr. Janez Žibert, Dr. Iztok Savnik, and Dr. Bojan Kuzma for their rewarding discussion and unselfish support.

My gratitude is also deserved by the members of the DIST research group from UP FAMNIT, who have provided help and friendship during my time in the group. My particular thanks go to Jernej, Tine, Andrej, Damjan, David, Pavel, Matjaž and Marko. My thanks go to the helpful and friendly members of department of mathmatics from UP FAMNIT and in particular to Klavdija, Cui, Martin, Boštjan, Ademir and Štefko.

A special thanks goes to the other colleagues and scientist in OR, EC, Economics, Organizations, Maths and Psychology for sharing with me their ideas, insights, views, projects, etc. during PhD studies. I would like to give thanks to Dr. William Cook, Dr. Peter Bosman, Dr. Marko Urh, Mr. Rado Pezdir and Dr. Milan Tuba.

I would also like to thank all my best friends, particularly Tadija, Pop, Neša, Crni, Irena, Tomaž, Peđa, Boštjan and Mlađa. You helped make my university life most enjoyable.

Thanks to my family in Belgrade: Dada, Daja, Olja, Dace, Peđa, Goran, Kaća, Jeca, Gaga, Boki, Sneža, Goga, Tića, Saška, and their families. I am very grateful for your support.

To the woman that is my life companion, my girlfriend Jasmina Simović. You bring laughter, tenderness and, above all, love to my life. Therefore, "thank you" are too small words to express gratitude.

Finally and most importantly, I thank my family for their love, and constant support and encouragement. I want to thank my father Gmitar Đorđević, my mother Slavica Đorđević and my brother Bojan Đorđević for the moral and spiritual harmony they represent for me.

Dedicated to my family!

Abstract

Grafted Genetic Algorithm and the Traveling Visitor Problem

In this PhD Thesis two related topics from theoretical computer science are considered. The first one considers Grafted Genetic Algorithms. The thesis analyzes separate, combined and partial performance of local searcher and genetic algorithm. On the Traveling Salesman Problem we examine the impact of hybridization a 2-opt heuristic based local searcher into the genetic algorithm. Genetic algorithm provides a diversification, while 2-opt improves intensification. Results on examples from TSPLIB show that this method combines good qualities from both methods applied and significantly outperforms each individual method. In tests we applied hybridization at various percentages of genetic algorithm iterations. We also studied at which iterations of the genetic algorithm to apply the hybridization. We applied it at random iterations, at the initial iterations, and the ending ones where the later proved to be the best.

The second topic considers the Traveling Visitor Problem. We consider the problem when a visitor wants to visit all interesting sites in a city exactly once and to come back to the hotel. Since, the visitors use only walking trails and pedestrian zones, the goal is to minimize the traveling visitor's walk. A new problem the Traveling Visitor Problem is then similar to the Traveling Salesman Problem with a difference that the traveling visitors, during its visit of sites, can not fly over buildings in the city, instead visitors have to go around these obstacles. That means that all Euclidean distances, like those in Euclidean TSP, are impossible in this case. The tested benchmarks are combined from three real instances made using tourist maps of cities of Koper, Belgrade and Venice and two instances of modified cases from TSPLIB. We compared two methods for solving the Traveling Visitor Problem. In all tested cases the Koper Algorithm significantly outperforms the Naïve Algorithm for solving the TVP.

ACM Computing Classification System: F.2.1, G.1.2, G.1.6, G.2.2, G.2.3.

Key words: Traveling Salesman Problem, Genetic Algorithms, Memetic Algorithms, Grafted Genetic Algorithms, Traveling Visitor Problem, Koper Algorithm, Naïve Algorithm.

Izvleček

Cepljeni genetski algoritmi in problem potujočega obiskovalca

V doktorski disertaciji sta obravnavani dve povezani temi s področja teoretičnega računalništva.

Tretje poglavje obravnava cepljene genetske algoritme. Disertacija analizira skupno in delno izvedbo lokalnega iskalca in genetskega algoritma. Na problemu trgovskega potnika (angl. Traveling Salesman Problem), smo preučevali vpliv hibridizacije 2-opt hevrističnega lokalnega iskalca v genetski algoritem. Genetski algoritem zagotavlja raznolikost (angl. diversification) , medtem ko 2-opt izboljša krepitev (angl. intensification). Rezultati primerov iz TSPLIB kažejo na to, da metoda združuje dobre lastnosti obeh metod in znatno prekaša vsako posamezno metodo. V testih smo uporabili hibridizacijo pri različnih odstotkih genetskih iteracij algoritma. Prav tako smo preizkušali, katere iteracije genetskega algoritma je potrebno hibridizirati. Hibridizacijo smo izvedli pri naključnih, začetnih in končnih iteracijah, za katere se je kasneje izkazalo, da so najboljše.

Četrto poglavje obravnava problem potujočega obiskovalca (angl. Traveling Visitor Problem). Problem nastane, ko si obiskovalec želi ogledati vse zanimive lokacije v mestu natanko enkrat in se po ogledu vrniti v hotel. Cilj je zmanjšati sprehod potujočega obiskovalca.

Problem potujočega obiskovalca je izpeljan iz problema trgovskega potnika , pri čemer velja pravilo, da obiskovalec izbira samo med potmi, ki jih je možno prehoditi. To pomeni, da so Evklidske razdalje (angl. Euclidean distance), kot jih poznamo v Evklidskem TSP, v našem primeru napačne. Obiskovalci uporabljajo sprehajalne poti in območja za pešce, ki so različno dolge. Te omejitve določajo težo povezav, ki povezujejo vozlišča v grafu. Za testiranje smo uporabili primere problema potujočega obiskovalca, ki smo jih izdelali iz uradnih turističnih zemljevidov za mesta Koper, Beograd in Benetke ter dveh spremenjenih primerov iz TSPLIB. Za reševanje problema potujočega obiskovalca smo primerjali dve metodi. V vseh testiranih primerih problema potujočega obiskovalca, algoritem koper občutno prekaša naivni algoritem.

ACM Computing Classification System: F.2.1, G.1.2, G.1.6, G.2.2, G.2.3.

Ključne besede: problem trgovskega obiskovalca, genetski algoritmi, memetični algoritmi, cepljeni genetski algoritmi, problem potujočega obiskovalca, algoritem koper, naivni algoritem.

Contents

List of Figures

List of Algorithms

List of Tables

Chapter 1

Introduction

I arrived to Koper, from Belgrade, on postgraduate studies of computer science, in 2007. Immediately, I had an urge to acquaint myself with the city in which I was to live for the next four years. I was drawn by the city center with its numerous sites - exactly 55 of them were listed in the official tourist map of Koper. Since doctoral studies were exhausting, I did not have much spare time to roam the streets of Koper. So I began to wonder whether I could devise a way of optimizing the tour through the city, by visiting all sites in as few steps as possible. We named the problem the *Traveling Visitor Problem* (TVP).

1.1 Traveling Visitor Problem

The Traveling Visitor Problem is a version of the *Traveling Salesman Problem*, (TSP) [5, 55, 60, 63, 66, 69, 75, 92, 99], with a difference that the traveling visitors, during their visit of sites, can not fly over the buildings in the city, instead visitors must go around these obstacles. This difference is demonstrated in Figure 1.1. This means that the Euclidean distances [57, 113], as we know them in the Euclidean TSP, are in this case impossible (direct edge from i to j in Figure 1.1). Visitors use the walking paths and pedestrian zones of variable length. These limits determine the weight of edges connecting the vertices in the graph.

Formally, the Traveling Visitor Problem is stated as: given a connected, weighted

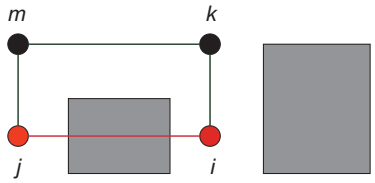

Figure 1.1: TSP and TVP

graph $G = (V, E, c)$, with a set of vertices $V = S \cup X$ and $S \cap X = \emptyset$, S belongs to interesting sites in a city, X belongs to crossroads in a city , a set of edges E, and a cost of traveling c. The goal is to find the shortest closed walk through all vertices of S, according to c in graph G, although we may walk through vertices from X.

Chapter 2 introduces the basic concepts of the Traveling Salesman Problem and related problems. Furthermore the algorithms for solving it, are described. This chapter, provides the knowledge necessary for further chapters of the dissertation to become understandable to the broad range of readers.

1.2 Traveling Salesman Problem

In the early 30's of the 20th century, the Austrian mathematician Karl Menger challenged the research community of that time to consider, from the mathematical point of view, the following problem: A traveling salesman has to visit exactly once each one of a list of n cities and then return to the home city. He knows the cost of traveling from any city i to any other city j. Thus, the question is which is the tour of least possible cost the salesman can take [100].

The instance of the TSP is formally defined on the complete graph G, with the set of vertices $V = \{v_1, v_2, ..., v_n\}$, for some integer n and by a cost function assigning a cost $c_{i,j}$ to the edge (v_i, v_j) for any i and j in G.

TSP can be viewed also as a permutation problem. Let P_n be a set of all permutations of the set $\{1, 2, ..., n\}$. Then the traveling salesman goal is to find a permutation $\pi = (\pi(1), \pi(2), ..., \pi(n))$ in P_n, that minimizes the quantity

$$\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(n)}, c_{\pi(1)}). \tag{1.1}$$

This quantity is referred to as the tour length since it is the length of the tour a salesman would make when visiting the cities in the order specified by the permutation π, returning at the end to the initial city.

TSP is one of the most important representatives of a large class of problems known as combinatorial optimization problems [67]. Since the TSP belongs to a class of *NP-hard* problems [76], an efficient algorithm for TSP probably does not exist. More accurately, such an algorithm exists if and only if the two computational classes P and NP coincide. From a practical point of view, it means that in general it is quite impossible to find an exact solution for any TSP instance with n nodes, that has a behaviour considerably better than the algorithm which computes all of the $(n-1)!$ possible distinct tours, and then returns the least costly one.

If we are looking for applications, a different approach can be used. Given a TSP instance with n nodes, any tour passing once through all cities is a feasible solution Algorithms that construct in polynomial time with respect to n feasible solutions are called heuristics [12,121]. In general, these algorithms produce solutions but without any quality guarantee as to how far is their cost from the optimal.

There exist two versions of TSP: the Symmetric TSP and the Asymmetric TSP. In the symmetric form, known as the STSP [73, 74, 93, 97], the cost distance between

nodes i and j is equal to the cost distance between nodes j and i ($c_{ij} = c_{ji}$). In the case of asymmetric TSP (ATSP) [14, 16, 18, 24], there is no such symmetry. In addition, there are many different variations of TSP which are described and explored in the literature and also a variations derived from everyday life. From the dissertation point of view, the most important applications of the TSP studied in the literature, are in more details presented in Chapter 2.

First steps in solving the TSP were consist of exact methods and heuristics. Exact methods like cutting planes [115] and branch and bound [27, 115], can optimally solve relatively small problems (depending on the size of n), while methods such as different variants of Lin-Kernighan algorithm [6, 46, 70, 79], and Concorde algorithm [3–5] are good for larger problems. Furthermore, some algorithms based on greedy principles such as nearest neighbour [63], and spanning tree [69] can be also used for solving a TSP. The above-mentioned methods for solving TSP result in exponential computational complexities. For that reason a new methods are required to overcome this shortcoming. These new methods include different kinds of heuristic techniques, nature based optimization algorithms, etc. Various creatures and natural systems, developed in nature, are interesting and valuable sources of inspiration for exploring and creating new methods for solving a TSP and variations of TSP. Let us count the most important of them. Evolutionary Algorithms [101, 106, 127, 136], Genetic Algorithms [43, 51, 52, 103, 109, 117, 126, 130, 134, 135, 139, 140], Memetic Algorithms [62, 87–89, 104, 113], Simulated Annealing [84], Ant Systems [39] and finally Grafted Genetic Algorithms [35], [38], [36], [37]. The latter is a type of hybrid genetic algorithms [43, 68, 94, 139] and they will be described and demonstrated in the thesis.

The first topic of the thesis, presented in Chapter 3, deals with the heuristic method called the Grafted Genetic Algorithm, through which we solve the Traveling Salesman Problem. In this chapter the aim is to show the quality of the solution and the running time of the grafted genetic algorithm when it is applied to the instances of the problems of symmetric TSP which are available on the Internet.

The second topic of the thesis, presented in Chapter 4, elaborates the Traveling Visitor Problem. This chapter describes a problem and examples of real cities and finally solves the instances of the problem by using new methods.

1.3 Research Objectives

Research objective of the thesis is to prove the following hypotheses 1 and 2. Proofs of hypothesis 1 and the hypothesis 2 are in the Chapter 3 and Chapter 4, respectively.

- Hypothesis 1: The method for solving of TSP that is made of two independent methods, genetic algorithm and 2-opt heuristic, outperforms each of the combined methods in terms of the quality of solution.
- Hypothesis 2: The quality of solution of a special method to the problem of a traveling visitor problem, outperforms algorithms for solving general TSP problem when they are used for solving traveling visitor problem.

Contributions to the science consist of the following results:

- Construction of the grafted genetic algorithm for solving the traveling salesman problem.
- Verification that the traveling salesman problem can be successfully solved using the grafted genetic algorithm.
- Construction of the special method for solving the traveling visitor problem.
- Construction of the real case instances of the traveling visitor problem for cities of Koper, Belgrade and Venice.
- Verification that all instances of the traveling visitor problem, which are solved using the special method, represent a very satisfactory solution.

The results of the thesis represents the contribution to bridging the gap between theoretical computer science and its application in practice. Also to better understanding and modeling of real problems in the economy, represented as the NP-hard problems from graph theory as well as a contribution to the optimization methods for solving these hard problems.

The results of this PhD Thesis are published in the following articles:

- M. Djordjevic, Influence of Grafting a Hybrid Searcher Into the Evolutionary Algorithm, *Proceedings of the 17th International Electrotechnical and Computer Science Conference, Portoroz, Slovenia* (2008), 115–118.
- M. Djordjevic, and M. Tuba, and B. Djordjevic, Impact of Grafting a 2-opt Algorithm Based Local Searcher Into the Genetic Algorithm, *Proceedings of the 9th WSEAS international conference on Applied informatics and communications, AIC 2009, Moscow, Russia* (2009), 485–490.
- M. Djordjevic, and A. Brodnik, Quantitative Analysis of Separate and Combined Performance of Local Searcher and Genetic Algorithm, *Book of Abstract of International Conference on Operations Research, OR 2011, Zurich, Switzerland* (2011), 130.
- M. Djordjevic, and A. Brodnik, Quantitative Analysis of Separate and Combined Performance of Local Searcher and Genetic Algorithm, *Proceedings of the 33rd International Conference on Information Technology Interfaces, ITI 2011, Dubrovnik, Croatia* (2011), 515–520.
- M. Djordjevic, A. Brodnik and M. Grgurovic, The Traveling Visitor Problem and Koper Algorithm for Solving It, accepted by *25th Conference of European Chapter on Combinatorial Optimization, ECCO2012, Antalya, Turkey.*
- M. Djordjevic, A. Brodnik and M. Grgurovic, The Traveling Visitor Problem and Algorithms for Solving It, accepted by *3rd Student Conference on Operational Research, SCOR 2012, Nottingham, UK.*

Chapter 2

Background

2.1 Traveling Salesman Problem

In this chapter we introduce the basic concepts of optimization and related combinatorial problems. Among those problems, probably best known is the Traveling Salesman Problem [5, 55, 60, 63, 64, 66, 69, 75, 92, 99], which we describe in this section in some detail. We illustrate the computational properties of the Traveling Salesman Problem which is closely related to many optimization problems arising in real world applications. We refine some graph theoretical approaches, which can be applied when attempting to tackle traveling salesman like problems. Traveling Salesman Problem can be stated as follows. Including his home town, a salesman wants to visit n cities and then return home. The finish line is to find a tour visiting each city exactly once while minimising the total distance traveled. The problem of finding a minimal tour is equivalent to finding an optimal ordering of the set of cities. The TSP is consequently often formulated as a permutation problem. Given a set of n cities $\{c_1, c_2, ..., c_n\}$ and for each pair of cities (c_i, c_j) a distance $d(c_i, c_j)$. The goal is then to find a permutation π of the cities which minimises the length of the tour given by:

$$\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(n)}, c_{\pi(1)}). \tag{2.1}$$

As mentioned before, the TSP has attracted the focus of researchers for decades. According to a survey on the history of combinatorial optimization by [124], the TSP was formulated as early as 1832 in a German manual for the successful traveling salesman before it was presented as a research problem by Menger in the 1930's. Since then, the TSPs simplicity on the one hand and the difficulty of finding optimal solutions on the other has affirmed it as a test bed for new heuristics and exact algorithms. But, the problem is not only a theoretical thing as there are many real life applications of the Traveling Salesman Problem and its variants [78, 114] some of which we will describe later in this chapter. Let us now introduce the Traveling Salesman Problem in mathematical terms as an integer programming problem. Take

up variables $x_{i,j}$ which are either equivalent to one if the city j is straightly visited after city i or zero otherwise. The Traveling Salesman Problem may be established as an integer program [83] as follows:

$$\begin{aligned} \text{Minimise } & \sum_{i=1}^{n}\sum_{j=1}^{n} d(c_i,c_j)x_{i,j}, \\ \text{Subject to } & \sum_{i=1}^{n} x_{i,j} = 1, \forall j = 1,2,...,n, \\ & \sum_{j=1}^{n} x_{i,j} = 1, \forall i = 1,2,...,n, \\ & \sum_{i\in I}\sum_{j\notin I} x_{i,j} \geq 1, I \subset \{1,2,...,n\} \text{ with } 1 < |I| < n, \\ \text{and } & x_{i,j} \in \{0,1\}, \forall i,j = 1,2,...,n, \end{aligned} \tag{2.2}$$

Here, our goal is to give the sum of all combination variables $x_{i,j}$ weighted by the lengths between the cities. This function is subject to a number of constraints. The constraints stated first and second inflict each city to be visited exactly once as a part of the tour. The constraints denoted third are the so called sub-tour avoidance constraints which are needed to guarantee that the solution represents a connected tour. The presentation of the Traveling Salesman Problem as an integer program shown above is not unique; there are various representations, see for example [112]. In the next subsections we will present some important notions for optimization problems and the Traveling Salesman Problem, together with a review of problems similar to the TSP.

2.1.1 Graph representation

Traveling Salesman Problem is ordinarily represented and considered as a graph theoretical problem [78,83]. This representation is suitable because many algorithms for the TSP are established on graph theoretical concepts. Here, I present some graph theoretical definitions needed to represent the Traveling Salesman Problem in graph theoretical terms.

An undirected graph $G = (V, E)$ consists of a finite set of vertices (or nodes) V and a finite set of edges E. Each edge $e \in E$ correspond to a set $e = \{u, v\}$ of two vertices $u, v \in V$. An edge $e = \{u, v\}$ is *incident* to vertices u and v and the number of edges incident to a vertex is the *degree* of the vertex. In the situation where there exists an edge $e = \{u, v\}$ between each pair $u, v \in V$ we speak of a *complete graph*.

Now, some concepts of the Traveling Salesman Problem will be presented. If $P = (\{u_1, u_2\}, \{u_2, u_3\}, ..., \{u_{k-1}, u_k\})$, where $\{u_i, u_{i+1}\} \in E$, then P is called a *walk*. If further $u_i \neq u_j$ for all $i \neq j$ it is called a *path*. A closed path, where $P = (\{u_1, u_2\}, \{u_2, u_3\}, ..., \{u_k, u_1\})$, i.e. a path plus the edge returning to the vertex it started from, is referred to as a *cycle*. Graphs are often classified according to their properties. For the TSP two important classes are the *Eulerian* and the *Hamiltonian*

graphs. A graph is called *Eulerian* if it contains an *Eulerian Tour*, which is a closed walk traversing every edge of the V. Similarly a graph is called *Hamiltonian* if there exists a cycle in G visiting every vertex of the graph, this cycle is called a *Hamiltonian Cycle*. For many applications of the TSP it is worthy to measure the qualities of a walk, path or tour in comparison to others and *weights* are associated with each edge of the graph. Given such a graph we speak then of a *weighted graph*. Since a walk, path or tour can be described by the set of edges traversed, P, its quality can be valuated by a weight or cost function $c : P \longrightarrow \mathbb{Q}$ mapping edge set P into the set of the rational numbers $\mathbb{Q}$. Let $c_{i,j}$ refer the weight of edge $\{i, j\}$, then the weight of an edge set P is then specified as the sum over all the edges of P:

$$\sum_{\{i,j\}\in P} c_{i,j}. \tag{2.3}$$

For a tour or a path the sum of weights of its edges is commonly specified to as its *length*. Let us now determine the symmetric TSP (STSP) [73, 74, 93, 97]. In the STSP the distance between nodes i and j is equal to the distance between nodes j and i. An instance of a TSP can be seen as a complete graph $G = (V, E)$ where the set of vertices V is given by the cities and edges in the graph with corresponding edge weights $c_{i,j}$ are given by the distances between cities. The Traveling Salesman Problem is then equal to the problem of finding a Hamiltonian Tour of minimal length in the graph G. For many applications it is useful to correlate a direction of the edges of a graph. Because the weights between vertices are not necessary symmetric, i.e. $c_{i,j} \neq c_{j,i}$, for some edges of G. This kind of graph is called *directed graph* or *digraph* and its edges are ordered 2-tuples of vertices, usually referred to as *arcs*. For arcs we have to distinguish between the arcs leading to and leading from a vertex. The concepts we summarized above for path, walk, tour, Hamiltonian and Eulerian tour can be modified easily to take the directions of the edges into account. The asymmetric traveling salesman problem (ATSP) [14, 16, 18, 24] is then similar to the symmetric TSP above, i.e. it is the problem of finding a Hamiltonian Tour of minimal length in a complete digraph). The *Euclidean TSP*, or planar TSP, is the TSP with the distance being the ordinary *Euclidean distance*. The Euclidean TSP [57, 113] is then a particular case of the metric TSP, since distances in a plane obey the triangle inequality.

2.1.2 Computational complexity

As was mentioned at the start of this section the Traveling Salesman Problem is a very hard combinatorial optimization problem. In order to evaluate algorithms according to their computational requirements the theory of *Computational Complexity* has been developed. Informally, the computational complexity deals with the number of computational steps an algorithm needs to solve an instance of a problem of size n. Since an algorithm can be described as a "step-by-step procedure" [76], the number of computational steps performed introduces a measure on the execution time required to solve an instance of a problem of a given size. However, as the number of steps an algorithm requires is often not only dependent on the size of the problem instance, but may also differ between instances of same size. Because, it is

not always straightforward to estimate the number of steps an algorithm needs for a given instance. In order to be able to analyse the complexity of an algorithm, a *worst-case analysis* is introduced. Here, the computational complexity is defined as the *maximum number of steps* an algorithm may require for any instance of a given size.

In summary, the performance of an algorithm is measured as the maximum number of steps it requires for any problem instance of size n denoted as a function of n. A widely accepted concept is that an algorithm is effective if its worst-case complexity is bounded by a polynomial function of instance size n while algorithms of exponential time complexity are usually considered ineffective or computationally expensive [76]. Although an exponential function may initially give smaller values than a given polynomial, there always exists a constant N such that for all $n \geq N$ the polynomial function takes smaller values than the exponential [83].

The Classes P and NP

The concept described above can be used to classify an algorithm as efficient or not depending on whether it has polynomial time complexity. Similarly a problem is considered easy or hard depending whether there exists a polynomial time algorithm for solving it. This means a problem is considered easy if there is an algorithm where in the worst case the number of steps of the problem of size n is bounded by a polynomial of n. The main purpose to investigate the computational complexity of decision problems is that there exist efficient methods for concerning the complexities of different decision problems. Decision Problems consist of an instance of a problem and a question to which the answer is 'yes' or 'no'. An example of a decision problem related to the TSP is the *Hamiltonian Cycle Problem* [83]:

Problem 2.1.1 *Instance: Graph $G = (V, E)$ Question: Does there exist a cycle in G passing through each vertex in V exactly once?*

As we described above, looking from the graph theoretical point of view, the TSP is equivalent to finding a Hamiltonian cycle of minimum length in a complete graph G. As a consequence, for the TSP decision problem the main question is not whether a cycle exists in a complete graph G but if there exists a cycle of length less than a given constant B. The TSP Decision Problem can then be stated as follows [83]:

Problem 2.1.2 *Instance: Given a complete weighted graph $G = (V, E)$ with non-negative edge weights ω_i, for $i \in E$ and a constant $B \geq 0$ Question: Does there exist a Hamiltonian cycle in G (or Traveling Salesman Tour) with edge sequence S where $\sum_{i \in S} w_i \leq B$?*

Even one may see that if there exists a polynomial time algorithm for the TSP there also exists a polynomial time algorithm for the TSP Decision Problem. On the other hand it is not so evident and after all true that the converse holds as well. A polynomial time algorithm for generating the optimal TSP tours by calling a subroutine that solves the TSP Decision Problem is presented by Johnson and Papadimitriou in [76]. Supposing that the TSP decision problem can be solved

in polynomial time, this algorithm, provides a polynomial time algorithm for the Traveling Salesman Problem. This is a type of polynomial time reduction [33]. Additionally the presence of this algorithm proves the following theorem [76]:

Theorem 2.1.3 *There exists a polynomial time algorithm for the TSP if and only if there exists a polynomial time algorithm for the TSP Decision Problem.*

This theorem indicate that when researcher is interested in the computational complexity it is adequate to restrain focus to the TSP Decision Problem. That's why the question of whether the TSP is a "hard" to solve problem is equivalent to whether the TSP Decision Problem is a "hard" problem. Let us now introduce the two classes of problems in the theory of computational complexity. The class P consists of all decision problems for which exists a polynomial time algorithm. There exists problems which are likely not in P for many combinatorial optimization problems. Nevertheless, it is not easily shown that certain problems are not in P even if no polynomial time algorithm is known. To specify the second class of algorithms we must first define the concept of *non-deterministic* algorithms.

In contrast to the deterministic algorithm, a non-deterministic algorithm can exhibit different behaviors on different runs. The class NP are composed of all decision problems which can be solved in polynomial time by an non-deterministic algorithm. It is obvious that $P \subseteq NP$. In spite of that, the question of whether $P = NP$ denote one of the major question of the computer science community. A lot of research has been spent on this challenge over the last five to six decades and the fact that no polynomial-time algorithm for specific problems in NP has been found, makes the equivalence of both classes very unlikely. Because of this, it is widely assumed that $P \neq NP$, even though this conjecture remains unproven to present date.

The Class NP-Complete

Very important subset of problems in NP is the class of $NP-complete$ problems. The most important property of NP-complete decision problems is that their computational status is directly connected to the relation between P and NP. The concept of NP-completeness was introduced in [26]. Regrettably, concerning the Traveling Salesman Problem, it has been shown that it belongs to the class of NP-complete problems. As Johnson and Papadimitriou comment in [76]:

> *This is our main negative result for the TSP; it is the strongest negative result one can hope to prove, short of establishing $P \neq NP$.*

In contrast to the consideration that the TSP is not NP- complete as it is not a decision problem. Before in the chapter we have mention a polynomial time algorithm for the TSP which directly lead to a polynomial time algorithm for the TSP Decision Problem and also its NP-completeness imply that $P = NP$. Problems which have this property are referred to as *NP-hard* problems.

2.1.3 TSP Applications

Applications of the TSP and its variations go way over the route planning problem of a traveling salesman and spans over several areas of knowledge including mathematics, computer science, operations research, genetics, engineering, and electronics. In the next three subsections I summarize, from the dissertation point of view, the most important applications of TSP studied in the literature.

Machine Scheduling Problems

Maybe the most studied application area of the Traveling Salesman Problem is sheduling and machine sequencing. A simple scheduling application can be described as follows. There are n jobs $\{1, 2, ..., n\}$ to be processed consecutive on a machine. Let c_i, be the setup cost required for processing job j instantly after job i. When all the jobs are processed, the machine is reset to its original state at a cost of $c_{i,j}$, where j is the last job processed. Therefore the machine sequencing problem is to find an order in which the jobs are to be processed such that the total setup cost is minimized. Obviously, finding a permutation π of $\{1, 2, ..., n\}$ that minimizes Equation 2.4 solves the problem.

$$c_{\pi(n)\pi(1)} + \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} \tag{2.4}$$

Looking from the practical point of view the jobs can be often clustered together. In this case the setup time, if any, between jobs within a cluster is relatively small compared to setup time between jobs in two different clusters. For example this can be one of typical scenario in assembly line. Assembling the similar products need minimal setup time. But if a different product needs to be assembled, the setup time may grow larger because it may require the new parts, tools etc. So the machine sequencing problem with such a specialized matrix of costs reduces to the *Clustered TSP*, this problem will be, in more detail, presented latter in desertation.

Let as now look at another scheduling problem-a *no wait flow shop* problem. There are n jobs each demanding processing on m machines in the order $1, 2, 3, ..., m$. No job is approved to have a waiting time between processing on two sequential machines. The goal is to find an optimum sequencing of jobs to be processed so that the total completion time is minimal. Applications of this kind of sequencing problems rise in a different situations, for more details one can looks at [44]. Furthemore, the no-wait flow shop problem is strongly NP-hard, but solvable in polynomial time when $m = 2$. In this particular case it reduces to the well known *Gilmore-Gomory* problem [63].

From an instance of the no-wait flow shop problem, it can be constructed an equivalent instance of TSP and so using the reduction proposed in [44]. Create a complete graph G on $n+1$ nodes, where node represent a job j, $1 \leq j \leq n$ and node $n+1$ represents both the start and the end of processing. The cost $c_{i,j}$ of edge (i, j) in G represents the additional schedule length if job j is the direct successor of job i in the schedule. So a minimum cost tour in G represent a schedule with minimum completition time. To finish the reduction, the values of $c_{i,j}$ must be identified. Let

$p_{j,k}$ be the processing time of job j on processor k, and also $1 \leq j \leq n$, $1 \leq k \leq m$. Then $c_{i,j}$ can be obtained using the equations [44]:

$$\begin{aligned} & c_{n+1,i} = \sum_{r=1}^{m} p_{ir}, i = 1, 2, ..., n, \\ & c_{ij} = max_{1 \leq k \leq m} \left\{ \sum_{r=1}^{k} p_{ir} + \sum_{r=k}^{m} p_{jr} \right\} - \sum_{r=1}^{m} p_{ir}, 1 \leq i, j \leq n, i \neq j \\ & c_{i,n+1} = c_{ii} = 0, i = 1, 2, ..., n. \end{aligned} \tag{2.5}$$

These applications that we considered so far reduce a given problem to the Traveling Salesman Problem.

For details of other research works on printed circuit board assembly that are relevant to the TSP, but goes beyond the scope and aim of the dissertation, we refer to [8, 63].

General Routing Problem

Given a connected, undirected graph $G = (V, E)$ consists of a finite set of vertices V and a finite set of edges E, a cost c_e for each edge $e \in E$, a finite set $V_R \subseteq V$ of required vertices and a finite set $E_R \subseteq E$ of required edges. The General Routing Problem (GRP) is the problem of finding a minimum cost route passing through each $v \in V_R$ and each $e \in E_R$ at least once [111].

The GRP contains several other important routing problems as special cases:

- The *Rural Postman Problem* (RPP) is obtained when $V_R = \emptyset$. For more details on this problem, we refer to [111].
- The *Chinese Postman Problem* (CPP) is obtained when $V_R = \emptyset$ and $E_R = E$. For more details on this particular problem, we refer to [25].
- The *Steiner Graphical Traveling Salesman Problem* (SGTSP) is obtained when $E_R = \emptyset$. The SGTSP has been discussed in [29]. Moreover, in [47] this problem was also called the *Road Traveling Salesman Problem* (RTSP).
- The *Graphical Traveling Salesman Problem* (GTSP) is obtained when $E_R = \emptyset$ and $V_R = V$. For more details we refer to [29].

Frequency Assignment Problem

If we look a communication network together with a set of transmitters, *the Frequency Assignment Problem* is to specify a frequency to each transmitter from a given set of available frequencies. The frequencies must satisfying some interference constraints. These constraints can be denoted by a graph $G = (V, E)$ in which each node denotes a transmitter. A non-negative integer weight c_{ij} is appointed for each arc (i, j) denoting the tolerance. Let $F = \{0, 1, 2, ..., R\}$ be a collection of available frequencies. A frequency assignment is then to assign the number $f(i) \in F$ to node

$i \in V$ such that $|f(i) - f(j)| > c_{ij}$ for all $(i, j) \in E$. If such an assignment exists, then it is called a feasible assignment. If R is enough large a feasible assignment is possible. Let us consider the following, let G^* be the complete graph obtained from G by adding edges of zero weight. Let $c'_{i,j}$ be the weight of edge (i, j) in G^* such that $c'_{ij} = c_{ij} + 1$. Finally, let $C'(H^*)$ be the sum of the weights of edges in a minimal cost Hamiltonian path in G^*. Then the TSP can be used to compute a lower bound for the frequency assignment problem [63].

2.1.4 More Variations of the TSP

Our discussion on next variations are confined to their definitions only and practical implementations for some variations. For more details on these problems, see the corresponding references cited. Also, the references I cite are not necessarily a paper in which the problem was introduced. Below I summarize, from the dissertation point of view, the most important variations of TSP studied in the literature.

The time dependent TSP: For each arc (i, j) of graph G, n of different costs $c^t_{ij}, t = 1, 2, ..., n$ are given. The cost c^t_{ij} represent the cost of traveling from city i to city j in some time period t. The goal is to find a tour $(\pi(1), \pi(2), ...\pi(n), (1))$, where $\pi(1) = 1$ represents the home location, which is in the time period zero, in graph G such that $\sum_{i=1}^{n} c^i_{\pi(i)\pi(i+1)}$ is minimized. The index $n + 1$ is equivalent to 1. For all arcs (i, j), if $c^1_{ij} = c^2_{ij} = \ldots = c^n_{ij}$ then this problem, the time dependent TSP, reduces to the TSP. For more details on this particular problem, we refer to [59].

Black and White TSP: The black and white TSP is a generalization of the Traveling Salesman Problem. In this problem, the set of nodes of G is partitioned into two sets, B and W The elements of set B are called black nodes and the elements of set W are called white nodes. A tour in G is feasible if the following two conditions are satisfied. The number of white nodes between any two sequential black nodes should not go beyond a positive integer I and the distance between any two sequential black nodes should not go beyond a positive real number R. The black and white TSP is to find a minimal cost feasible tour in graph G. Applications of this problem involve design of ring networks in the industry of telecommunication [63]. Furthemore, a variation of this problem known as TSP with replenishment arcs, has been applied in the air-line industry. The TSP with replenishment arcs has been discussed in [98].

The delivery man problem: The delivery man problem (DMP) is also known as the *minimum latency problem* [11] and *the traveling repairman problem* [53]. Let H be a tour in G and v_1 be a starting node. For each vertex v_i of G, define the *latency* of v_i with respect to H denoted by $L_i(H)$, is the total distance in H from v_1 to v_i. The delivery man problem is then to find a tour H^* in graph G such that $\sum_{i=1}^{n} L_i(H^*)$ is as small as possible. The DMP is strongly NP-complete. Additionally it can be verified that this problem is a special case of the time dependent TSP. For more details on this problem, we refer to [63].

Clustered TSP: In this vatiation of TSP, the node set of G is partitioned into clusters $V_1, V_2, ..., V_k$. Then the clustered TSP [65] is to find a minimum cost tour in graph G with the constraint that nodes within the same cluster must be visited consecutively. By adding a large cost M to the cost of each inter-cluster edge this

problem can be reduced to a Traveling Salesman Problem.

Generalized TSP (GTSP): As in the case of clustered TSP, let $V_1, V_2, \ldots, V_k$ be a partition of the node set of graph G. In a GTSP, the goal is to find a shortest cycle in graph G which goes through exactly one city from each cluster $V_i, 1 \leq i \leq k$. If $|V_i| = 1$ for all i, GTSP is the same as Traveling Salesman Problem. Using the reduction described in [110] we show that GTSP can be reduced to a TSP for arbitrary $|V_i|$. WLOG, we can assume that graph G is a digraph and the partitions are numbered in such a way that $|V_i| \geq 2$ for $1 \leq i \leq r$ and $|V_i| = 1$ for $r+1 \leq i \leq k$. For any $i \leq r$ let $V_i = \{i_1, i_2, ..., i_n\}$. For each arc $e \in E$, consider a new cost d_e defined as follows:

$$d_{i_j i_{j+1}} = -M, j = 1, \ldots, n_i \text{ with } n_i + 1 \equiv 1, i = 1, ..., r. \tag{2.6}$$

This guarantee that if a minimal TSP tour in graph G enters a cluster V_i through node i_j, it visits nodes of V_i in the order $i_j, i_{j+1}, ..., i_{n_i}, i_1, ..., i_{j-1}$ and leaves the cluster V_i from the node i_{j-1}. We now want to interpret this outcome equivalent to a GTSP tour entering and leaving the cluster V_i by visiting node i_j. To duplicate this fact, we make the new cost of arcs going out of i_{j-1} represents to the original cost of arcs leaving i_j. To be more precise look at Equation 2.7.

$$d_{i_{j-1}p} = c_{i_j p}, p \notin V_i, j = 1, 2, ..., n_i \text{ with index } 0 \equiv n_i, i = 1, ..., r \tag{2.7}$$

and $d_{uv} = c_{uv}$ for all other edges. From a minimal solution to the TSP on graph G with the modified costs d_e for $e \in E$, a minimal solution to GTSP can be recovered. For more details on this problem, we refer to [62, 79].

The MAX TSP: In contrast to the TSP, the objective in the MAX TSP is to find a tour in graph G where the total cost of edges of the tour is maximum. This problem can be solved as a TSP by just replacing each edge cost by its additive inverse. If problem requires that the edge costs are non-negative, a large constant could be added to each of the edge-costs. These replacing will not change the optimal solutions of the problem. The MAX TSP has been discussed in [63].

Traveling Tourist Problem: A tourist wishes to see all monuments in a city, and so must visit each monument or a neighbour thereof. Furthemore, it is assumed that a monument is visible from any of its neighbours. The edges therefore represent lines of sight. The resulting walk will therefore visit a subset of all nodes in the graph G. For the Traveling Tourist Problem we refer to the [95].

The bottleneck TSP: In this variation of TSP the objective is to find a tour in graph G such that the largest cost of edges in the tour is as small as possible. A bottleneck TSP can be described as a TSP with exponentially large edge costs. For more details we refer to [63].

TSP with multiple visits (TSPM): In this problem the objective is to find a routing of a traveling salesman. Salesman starts at a given vertex of graph G, visits each vertex at least once and comes back to the starting vertex in such a way that the total distance traveled is minimized. The TSPM can be transformed into a TSP by replacing the edge costs with the shortest path distances in graph G. In the lack of negative cycles, shortest path distances between all pairs of vertices of a graph G

can be computed using efficient algorithms [1]. If graph G contains a negative cycle, then TSPM is unbounded. For more details on TSPM look in [63].

2.1.5 All-Pairs Shortest Paths

In this section, the problem of finding shortest paths between all pairs of vertices of graph is considered. Given a weighted, directed graph $G = (V, E)$ with a weight function $\omega : E \to \mathbb{R}$ that maps edges to real valued weights. The objective is to find, for every pair of vertices $u, v \in V$, the shortest path from u to v. In addition the weight of a path is the sum of the weights of its edges. The all-pairs shortest-paths problem, can be solved, by running a single-source shortest-paths algorithm $|V|$ times, once for each node as the source. In some particular case where all edge weights are nonnegative, *Dijkstra's algorithm* [28], can be used for solving the problem. If the linear-array implementation of the min-priority queue is used, then the running time is $O(V^3 + VE) = O(V^3)$. Furthemore, the binary min-heap implementation of the min-priority queue generate a running time of $O(VE\ lg\ V)$, which is an improvement if the graph is *sparse*. If the graph contains the negative-weight edges, then Dijkstra's algorithm can not be used. Alternatively, the slower Bellman-Ford algorithm can be run, once from each node. The resulting running time is then $O(V^2E)$, which on a dense graph is $O(V^4)$. In addition, for solving the all-pairs shortest paths problem on sparse graphs the *Johnson's algorithm* is used, for more details see [28]. In contrast to the single-source algorithms, which assume a distance-list representation of the graph, the *Floyd-Warshall algorithm* uses a distance-matrix representation.

The Floyd-Warshall Algorithm

The Floyd-Warshall algorithm (FW) is a simple and extensively used algorithm for computing the shortest paths between all pairs of vertices in an edge weighted directed graph $G = (V, E)$. The algorithm runs in $\Theta(V^3)$. The Floyd-Warshall algorithm produce the correct result as long as no negative cycles exist in the input graph. The FW algorithm take into consideration the intermediate nodes of a shortest path, where an intermediate node of a simple path $p = \langle v_1, v_2, \ldots, v_l \rangle$ is any node of p other than v_1 or v_l, that is, any node in the set $\{v_2, v_3, \ldots, v_{l-1}\}$. The Floyd-Warshall algorithm depend on the following observation. During assumption that the nodes of G are $V = \{1, 2, \ldots, n\}$, consider a subset $\{1, 2, \ldots, k\}$ of nodes for some k. For any pair of nodes $i, j \in V$, consider all paths from i to j whose intermediate nodes are all drawn from $\{1, 2, \ldots, k\}$, and let p be a minimal-weighted path from between them. The Floyd-Warshall algorithm take advantage of a relationship between path p and shortest paths from i to j with all intermediate nodes in the set $\{1, 2, \ldots, k-1\}$. Furthemore, this relationship depends on whether or not k is an intermediate node of path p.

The running time of the Floyd-Warshall algorithm shown in Algorithm 1 is appointed by the triply nested **for** loops of lines 2-4. Furthemore, each execution of line 5 takes $O(1)$ time, so the algorithm runs in time $\Theta(n^3)$. In addition, the constant hidden in the Θ-notation is small. Because of that the Floyd-Warshall algorithm is pretty efficient for even modest sized input graphs.

Algorithm 1 Floyd-Warshall

```
1: procedure FLOYD-WARSHALL(V,W)
2:     for all k ∈ V do
3:         for all i ∈ V do
4:             for all j ∈ V do
5:                 W_ij := min(W_ij, W_ik + W_kj)
6:             end for
7:         end for
8:     end for
9: end procedure
```

2.2 TSP Heuristics

In this section, different heuristic approaches that have been recommended in the literature for the Traveling Salesman Problem will be examined. Heuristics correspond to approximation algorithms with objective to find the near optimal solutions quickly rather than the best solution there is to a given problem. This section introduces the basic concepts of heuristic approaches for the Traveling Salesman Problem and give some of the theoretical results that have been examined in the literature. The scope of the section is mainly concentrate on the classical constructive, local search approaches and by nature inspired techniques and their extensions. For the Traveling Salesman Problem these diverse approaches represent the state-of-the-art when the objective is to find satisfactory solutions quickly. The expression heuristic is interpreted in the Oxford dictionary as:

> *a method for solving problems by learning from past experiences and investigating practical ways of finding a solution*

The goal of heuristic approaches is to find satisfactory or close to optimal solutions quickly instead of finding a global optimal solution. Heuristic approaches for the Traveling Salesman Problem have been presented in several surveys. For the comprehensive overview of the heuristic tehniques, we refer a reader of the dissertation to the surveys [74, 78] and to a book on the TSP and its variants [63].

2.2.1 Tour Quality

The classic measure of the quality of a tour length is the gap to the optimal tour length. This gap is commonly demonstrated as the pass over optimum. For instance I this gap can be formally shown as:

$$\frac{A(I) - OPT(I)}{OPT(I)} \times 100\%. \tag{2.8}$$

Luckily, the standard test beds, such as Reinelt's TSPLIB [122], provides a wide set of instances which also includes the optimal tour length for all test instances. The optimal tour lengths are usually unknown for instances of sizes or structure

that cannot be solved to optimality in a reasonable amount of time. So a different reference point is needed. Those reference points present one of values which are close to the optimal tour length. Furthemore, the reference point of choice is as a consequence commonly a lower bound on the optimal solution.

The standard lower bound for the TSP is the Held-Karp lower bound [69]. The Held-Karp lower bound on the optimal solution corresponds to the solution of a linear programming relaxation [23] of the standard integer programming formulation of the TSP. This imply that the integer constraints of the integer programming problem are substitute with bounded variables. Furthemore, the resulting linear programming problem [132] is then solved to optimality by using an algorithm such as the simplex method [32]. Even if the resulting linear program consist an exponential number of subtour constraints it can be solved in polynomial time because there exists a polynomial time "seperation oracle" for the subtour constraints based on calls to a max-flow algorithm [80]. Experimental results which is presented in [63] show that this lower bound is very close to the optimal length in most cases.

2.2.2 Tour Construction Algorithms

The objective of tour construction approaches is to build a tour for an instance of traveling salesman problem from scratch. This is commonly achieved by constructing a tour following some construction rule. Determinations during the construction of a tour are mostly made in a greedy fashion. The objective of these methods is an immediate gain instead of looking ahead. There are many constructive heuristics which have been suggested for the TSP. For a comprehensive overview of tour construction approaches for the TSP look in [10, 123]. For an extensive empirical performance testing of the tour construction approaches for the symmetric TSP, especially the Euclidean TSP see [63, 123].

The tour construction approaches described in the next subsections were chosen because of their suitability to symmetric especially Euclidean instances of Traveling Salesman Problem. Two basic and from the dissertation point of view important tour construction approaches are the *Nearest Neighbour* and the *Insertion* heuristics.

Nearest Neighbour Heuristic

The Nearest Neighbour algorithm is one of the most intuitive heuristic algorithms for the Traveling Salesman Problem. The salesman starts at an arbitrary chosen node and then successively travels to the nearest node he has not yet visited. When all nodes have been visited, salesman returns to the node he started from. The pseudocode of this algorithm for a set of nodes i.e. cities $C = \{c_1, c_2, ..., c_n\}$ is shown in Algorithm 2. The Nearest Neighbour algorithm has a computational complexity of $O(n^2)$ which can be reduced to $O(n \log n)$ for geometric instances [10]. The best performance ratio known for the instances of TSP satisfying the triangle inequality is given by $\frac{NN(i)}{OPT(i)} \leq 0.5\,(\lfloor \log_2 n \rfloor + 1)$, see [74], and therefore growing in n. Nevertheless, in the case where the triangle inequality is not satisfied one can expect the

performance ratio to be even worse. In the literature exists various variants of the Nearest Neighbour approach with objective to improve its performance. One such variant is the Double Ended Nearest Neighbour [123], where the tour is built from both ends of the current sequence. Another variant is so-called Randomised Nearest Neighbour, where the city to be visited next is chosen randomly from a set of nearest neighbours [63].

Algorithm 2 Nearest Neighbour

1: **procedure** NEARESTNEIGHBOUR(C)
2: Select arbitary city c_j, set $k = j$ and $C = \{c_1, ...c_n\} \setminus c_j$
3: **while** $(C \neq 0)$ **do**
4: Determine $c_l \in C$ with $d(c_k, c_l) = \min_{c_j \in C}(d(c_k, c_j))$
5: Add c_l to the tour by connecting c_k to c_l and set $c \leftarrow C \setminus c_l$ and $k = l$
6: **end while**
7: Connect c_k to starting city c_j
8: **end procedure**

Insertion Heuristic

For the insertion heuristics a different construction rule is followed. The algorithm start with a subtour consisting of one or two cities. Then it successively add cities to the current subtour. This adding is followed by some selection criteria. The pseudocode of this approach is presented in Algorithm 3.

Algorithm 3 Insertion

1: **procedure** INSERTION
2: Select a starting tour through k cities $T = \{c_1, c_2, \ldots, c_k\}$
3: **repeat**
4: Select a city c_i with $c_i \notin T$ following some criteria
5: Insert city c_i into the current subtour T
6: **until** $c_i \notin T, i = 1, 2, ..., n$
7: **end procedure**

From the description of the algorithm three questions arise: From which city or cities to start the heuristic? Which city to insert next? Where to insert it into the current subtour? For each of these questions various decision rules have been proposed in the literature. The set of cities which construct the starting subtour is commonly chosen at random and consists of one, two or three cities. For Euclidean problems other approaches such as starting the heuristic from the convex hull of the problem have been also suggested [123]. The last two questions, to which city to insert next and where to insert it are tightly related. Some selection rules are:
1. Nearest Insertion: Insert the city which is not yet part of the tour and is nearest to any city already part of the tour,
2. Farthest Insertion: Insert the city that is not yet part of the tour whose minimal

distance to a city, already part of the tour, is maximal,
3. Cheapest Insertion: Insert the city which is not yet part of the tour whose insertion results to the minimal increase in tour length of the current subtour.
4. Random Insertion: Select the city to be inserted randomly out of the set of cities that are not yet part of the tour. For an empirical analysis of Insertion Heuristics with different insertion criteria look in [123]. Nearest insertion, farthest insertion and random insertion can be implemented to run in time $O(n^2)$. Cheapest insertion has time complexity $O(n^2 \log n)$ and so is computationally more expensive.

2.2.3 Local Search Algorithms

Local search algorithms for the Traveling Salesman Problem are built on simple tour adjustments. A local search algorithm is construted from operations called moves which are used to transform one tour to another. The local search is actually a neighborhood search process where each tour has an associated neighborhood of tours. The local search algorithm repeatedly moves to a better neighbor as far as no better neighbors exist. These moves which have been proposed for the TSP can be generaly divided into operators of node exchange, node insertion, and edge exchange. The node exchange operator work in such a way that it exchanges two nodes in the sequence. On the other hand the node insertion operators work by deleting a node from a tour and inserting it at another position in the tour. The edge exchange operators exchange the edges in the tour and will be described in detail in after section. For the example of the TSP a tour t^*, is called locally optimal when all other tours in its neighbourhood are at least as long as t^*. To identify a local optimum, the neighbourhood of an initial tour is searched for a solution of better quality. When a new best solution s', has been found, it is accepted and its neighbourhood $N(s')$ is explored. This process is repeated as far as no more improvements can be found. A pseudocode using above notifications of solutions and neighbourhoods is given in Algorithm 4.

Algorithm 4 Local Search

```
1: procedure LocalSearch
2:     Create feasible Solution s
3:     Choose a neighbourhood function N
4:     repeat
5:         Search neighbourhood N(s)
6:         if (s' with f(s') < f(s) found) then
7:             Set s = s'
8:         end if
9:     until no feasible lower cost solution s‘ is found
10: end procedure
```

The objective of local search algorithm is to consecutively apply improving moves to the solution in order to explore the neighbourhood introduced by N. For the TSP this neighbourhood is commonly based on simple tour modifications.

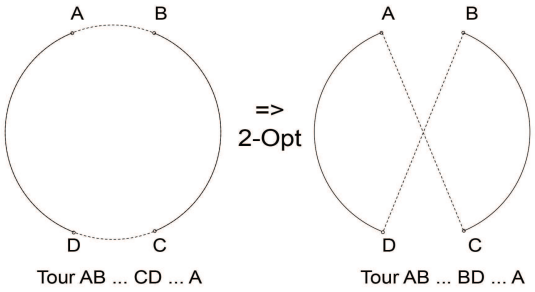

Figure 2.1: Edge removal and reconnection of 2-opt algorithm

k-opt Local Search

Among simple local search algorithms, the most famous are k-exchange neighbourhood which is an example of edge exchange algorithm. The k-exchange neighbourhood is for the Traveling Salesman Problem commonly referred to as k-opt. For the TSP this local search can be defined as follows. Let S corresponds to the set of all tours of a TSP instance. Let us also introduce a metric p between tours in S, which measures the distance between tours with the number of edges not mutual to both, i.e. the number of edges which represent part of tour T but not part of tour T'. The k-opt local search can then be stated by:

$$N_k(T) = \left\{T' | p(T, T') = k, T' \in S\right\}.$$

From the above definition the reader can see that this statement introduces a whole set of search neighbourhoods which is parameterised by k. Usually, the higher the k of a k-opt local search, the better the resulting tours. In spite of this, since the neighborhood size grows exponentially with k, only small k appear to be practical.

2-opt

The 2-opt local search illustrates the simplest of the k-opt local search algorithms for the TSP. The 2-opt algorithm was first proposed in [30], even though the basic move had already been suggested in [49]. This move deletes two edges, as a consequence the tour is aparted into two segments, and then algorithm reconnects those segments in the other possible way. A schematic illustration of this edge removal and reconnection approach is presented in Figure 2.1.

Just as is shown in the figure, this operation is equal to a tour segment reversal. The order of the tour segment $B, ..., C$ is reversed with respect to $C, ..., B$. For the Euclidean TSP, 2-opt local search removes the *crossings* of edges in the tour. For symmetric TSPs a 2-opt move usually produce an improvement to the current tour if $d(A, C) + d(B, D) < d(A, B) + d(C, D)$. There exist two different representations of a 2-opt move, i.e. the move presented in the Figure 2.1 is equivalent to the move where the tour segment between D and A is reversed. Since for a 2-opt move to

denote an improvement either $d(A,B) > d(B,D)$ or $d(C,D) > d(A,C)$ or both must hold. Because of this the attention can be concentrate on moves satisfying $d(A,B) > d(B,D)$. This imply that once A and B are fixed we can limit the search for an improving move on cities D that are closer to B than A.

So to take advantage of this feature Steiglitz and Weiner [128] introduce a data structure to specify suitable candidates quickly. And so, storing for each city a list of the remaining cities sorted with increasing distances. When seeking for a 2-opt move, one has to start at the beginning of the list of B, move forward through the list observing its members until $d(A,B) \leq d(B,D)$. The difficulty of this approach is that creating the lists is of time complexity $\Theta(n^2 \log n)$ and requires space quadratic in n. As a consequence in practice this list is commonly restricted to only k-nearest neighbours for each city with fixed k. Even if the resulting tour may not be locally 2-optimal, as not all possible moves are considered. In the practice, in general, only a small loss of tour quality is reported [74].

The Lin-Kernighan Algorithm

For over 30 years the world's champion heuristic for the Traveling Salesman Problem was usually recognized to be the local search algorithm of Lin and Kernighan (LK) [96]. This algorithm is both a generalization of the 2-opt local search algorithm and an outgrowth of ideas the same authors had previously applied to the graph partitioning problem. The fundamental idea of the LK heuristic is to build up complex moves by combining simple submoves to replace a variable number of edges. Because of this property this tehnique is also called a variable depth k-opt. The submoves commonly employed are 2-opt and 3-opt moves. The LK heuristic can be described as follows. In each step, the tour is broken up at one node forming a 1-tree (a spanning tree with an extra edge). This 1-tree can be with ease transformed into a possible TSP tour by breaking up one edge of the degree-3 node and connecting the two degree-1 nodes. Consequently the heuristic performs sequential changes of edges until no further exchanges are possible or until the best k-change in an iteration is found. A more comprehensive description of the LK algorithm would go way beyond the scope of a dissertation and can be found in the paper by Lin and Kernighan [96]. The original Lin-Kernighan heuristic was suggested for the symmetric TSP only.

A main drawback of the LK algorithm, besides the high effort needed for its implementation, is its rather long running time. Because of this, several improvements to the original algorithm have been made. In the paper [123] Reinelt, describes a variant of LK algorithm using the segment reversals and node insertion. In this variant a special 3-opt move consists of a single city to enlarge the Lin-Kernighan neighbourhood. Some different approaches can be found in [138], and in [120]. In this two papers the search neighbourhood is the so called flower transition, firstly suggested by Glover in [56] where the base is a 1-tree consisting of a cycle and a path attached to it. A effective variant of LK heuristic based on a sequence of 5-opt moves instead of 2-opt segment reversals was suggested in 2000 by Helsgaun [70]. The estimation of these move sequences is accomplished by considering very small candidate lists which are determined in a very complex manner. Other approaches

and variants of the LK heuristic have try to beat problems produced by very large TSP instances. Otherwise, to make the Lin-Kernighan searches perform better on the problematic instances of TSP [6]. Even though the variant of LK heuristic proposed by Helsgaun [70] is commonly the most successful one, when only a tour quality is considered. Otherwise, it is hard to say which one of these variants is the most effective. For a comparison of the Lin-Kernighan implementations we refer a reader of a dissertation to a Johnson and McGeoch work on heuristics for the symmetric Traveling Salesman Problem [75].

2.2.4 Nature Inspired Algorithms

The area of nature-inspired computing has grown in popularity over the last fifty years. Many of the nature-inspired algorithms currently in use are being applied to a wide range of problems, among them are of course ones tackling the Traveling Salesman Problem. The term *nature* is used to refer to any component of the universe which is not a product of planned human design. The nature-inspired algorithms are in the category of the metaheuristic algorithms, where little or no problem particular information is used in the design of the algorithm. Beyond all doubt the most important contribution to biology was made by Charles Darwin with his Theory of Evolution by Natural Selection. As Chiong comments in [22] Observing the achievements of animals, Darwin writes:

> *Why, if man can by patience select variations most useful to himself, should nature fail in selecting variations useful, under changing conditions of life, to her living products.*

If we restrict our attention to the biological part of nature, we can highlight some of the useful properties. In nature, managing the trade-off between solution quality and time is basic to survival. A likewise trade-off is made when using a heuristic to solve the optimization problems. The fittest individuals are those with supreme problem-solving feature. It is these problem-solving features which have been the source of inspiration for many nature-inspired tehniques. We will now represent several metaheuristic examples of nature-inspired algorithms for optimization problems, above all those tehniques which is used for tackling the Traveling Salesman Problem.

Simulated Annealing

Simulated Annealing (SA) was introduced by Kirkpatric et.al. in [85]. The SA optimization method builds on a similarity derived from physics processes where a low energy state of a solid is inquired by an annealing procedure. The similarity with combinatorial optimization occur when the optimal solution to a given combinatorial optimization problem corresponds with the lowest energy of the solid. Therefore, a solution to a problem is somehow perturbed and a neighbor solution is accepted with probability according to a Boltzmann distribution $e^{\frac{-\Delta E}{k*T}}$. Here ΔE correspond to the

difference in quality between the current solution and the perturbed one. Furthemore, k is a scaling parameter and T correspond to the temperature of the process. The higher the temperature the greater is the probability of acquiring a perturbed solution. When the temperature is low, improving moves will be privileged. By reducing T using an "annealing schedule" it is possible to simulate the freezing process. The pseudocode of the SA is shown in Algorithm 5:

Algorithm 5 Simulated Annealing

```
1: procedure SimulatedAnnealing
2:     t = T(0), n = 1
3:     best solution s_best = S
4:     while (Termination Criterion Unfulfilled) do
5:         Generate s' ∈ N(s)
6:         /* we obtain a neighbor of s using the move operator N() */
7:         Δf = f(s) − f(s')
8:         /* we compute the difference in fitness */
9:         if ((Δf ≤ 0) or (e^(−ΔE/(k*T)) > random[0,1])) then
10:            s = s'
11:            /* we accept the perturbed move if it is better than the current */
12:            /* or if the Boltzmann criterion is satisfied */
13:        end if
14:        if (f(s) > f(s_best)) then
15:            s_best = s
16:        end if
17:        t = T(n)
18:        /* we apply the annealing schedule for time n */
19:        n = n + 1
20:    end while
21:    Return s_best
22: end procedure
```

The key to the efficiency of SA in solving a specific combinatorial optimization problem stands in the definition of the the annealing schedule and move operator. Since the first appearance of SA, the Traveling Salesman Problem has served as a test problem, considered in the original paper of Kirpatrick [85]. For these particular approach the SA neighbourhood was 2-opt local searcher. In addition, Boese's PhD dissertation [13] comprehensively analyzes the optimal annealing schemes for the TSP.

Ant Colony Optimization

A recent and increasingly popular metaheuristic approach inspired by nature is the Ant Colony Optimization (ACO) or Ant Algorithms introduced in 1997 by Dorigo. ACO [39] presents a population based approach that was inspired by the behavior of real ants in nature. The main idea of this approach trace the estimation

that ants in nature appear very effective in finding the shortest paths to a food source. The ants are capable to find the shortest paths through communication transmitted by leaving pheromone trails while exploring for food. For the Traveling Salesman Problem, ACO represents a constructive approach which is frequently updated until some stopping condition is reached. Starting with n ants, each located at a different city, the ants consecutively move along the edges to create feasible tours. Determination on which city to visit next are made in a probabalistic manner based on the pheromone trail left at earlier investigation. After each iteration the pheromone trail is updated depositing a higher amount of pheromone at edges used in the shortest tours.

Several improvements to this basic approach have been proposed in the literature, especially the conjunction with local search to speed up the search process [129]. The computational results described in the literature range from very poor to moderate quality in [40] to solutions of very good quality presented in [129]. For more knowledge on ACO for the TSP and also to some other combinatorial optimization problems we refer to [40, 129].

Evolutionary Algorithms

Evolutionary Algorithms (EA) simulate the process of biological evolution in nature. These are search methods which are motivated by natural selection and survival of the fittest from biological world. EA performs a search using a population of solutions. Each iteration i.e. generation of an EA includes a competitive selection among all solutions in the population. This selection results in survival of the fittest and erasure of the poor solutions from the population. Recombination is performed by exchanging parts of a solution with another one. This proces forms the new solution that may be better than the previous ones. Furthermore, a solution can be mutated by manipulating a part of it. The evolutionary operators, recombination and mutation, are used to evolve the population towards areas of the space in which good solutions exist. Four major evolutionary algorithm classes have been introduced during the last 50 years: genetic programming is a computational method, which was proposed Koza [86], Evolutionary Strategies (ES) developed by Rechenberg [118], Evolutionary Programming (EP) introduced by Fogel [50], and Genetic Algorithms (GA), proposed by Holland [71]. The general template of an EA is shown in Algorithm 6. All mentioned variants of evolutionary algorithm (GA, EP, and ES) are special cases of this scheme.

For the Traveling Salesman Problem the evolutionary algorithms may be shortly outlined as follows. Starting from a population of k individuals i.e. tours, select k' different individuals for mutation and parents for mating. Perform mutations on the selected individuals and the mating recombination. In recombination the information of two parent individuals are combined in order to create an offspring individual. Then select k existing individuals of the new population succeeding the selection strategy. This procedure is consecutively applied for some iterations i.e. generations until some termination criteria is met. For further reading on Evolutionary Algorithms we refer to [7, 106, 127].

Algorithm 6 Evolutionary Algorithms

```
1: procedure EA
2:     t := 0
3:     initialize population (P(0))
4:     evaluate (P(0))
5:     repeat
6:         P' := select for variation (P(t))
7:         recombine (P')
8:         mutate (P')
9:         evaluate (P')
10:        P(t + 1) := select for survival (P(t), P')
11:        t := t + 1
12:    until terminate = true
13: end procedure
```

Genetic Algorithms

Genetic Algorithms were introduced by Holland in the 1970s [71], and well researched by many authors due present date [43, 51, 52, 103, 109, 117, 126, 130, 134, 135, 139, 140]. These techniques are adaptive search methods based on the instrument of natural selection and the survival of the fittest concept. A comprehensive introduction to Genetic Algorithms is given in Goldberg's book [58]. The main inspiration behind GA is to start with randomly created initial solutions and implement the survival of the fittest strategy to develop the better solutions through iterations i.e. generations. A Genetic Algorithm process includes initial population generation, fitness evaluation, chromosome selection and applying the genetic operators for reproduction, recombination and mutation. The pseudocode of a GA can be seen in Algorithm 7.

Algorithm 7 Genetic Algorithm

```
1: procedure GA
2:     Randomly generate an initial population P(t)
3:     while (Termination Criterion Unfulfilled) do
4:         Compute the fitness f(p) ∀p ∈ P(t)
5:         According to f(p) choose a subset of P(T), store them in M(t)
6:         Recombine and mutate individuals in M(t), store results in M'(t)
7:         Generate P(t + 1) by selecting some individuals from P(t) and M'(t)
8:         t = t + 1
9:     end while
10:    Return best p ∈ P(t − 1)
11: end procedure
```

In designing a Genetic Algorithm, how to encode a search solution is a basic and key issue [21]. Many optimization operators for Traveling Salesman Problem were proposed by Goldberg [58]. A usually used encoding strategy is transposition

expression [119]. In the transposition expression encoding strategy, each city of the TSP is encoded as a gene of the chromosome. This encoding include the constraint that each city appears once and only once in the chromosome. Transposition expression is the most nature expression for TSP which based on the order of tour, on the other hand such a procedure may leads to infeasible tour after traditional crossover operator. This is a usual occurrence for TSP. Even if feasibility can be maintained in many ways by some repair algorithms, such algorithms can spend a substantial amount of time and can inhibit convergence [119].

To overcome a possible creation of infeasible tours another encoding method was introduced. It is the Random Keys encoding [9] which is introduced by Bean. In random keys encoding a random numbers encode the construction of the solution. Such representation guarantees that feasible tours are preserved during the application of genetic operators. In the GA, the crossover and mutation are two of most important factors for the success of the algorithm. Crossover is a recombining operator that takes two individuals, and cuts their chromosome strings at a number of chosen positions. The sub-segments produced by that cuts are then mixed and reconnected to produce a whole chromosome with features of the two parents. Therefore, the basic role of recombination is of information exchange between successful solutions.

Numbers of different recombination operators have been proposed in the literature to solve the TSP using a GA. The partially mapped crossover [58], linear order crossover [34] and order based crossover [9, 58] are the usually used recombination tehniques in the TSP context. Except the usually used recombination strategy, many different recombination operators are proposed for the TSP, for example: sub-tour crossover [137], edge map crossover [51], distance preserving crossover [103], generic crossover [104], NGA [77], EAX [107], GSX [108], heuristic based crossover [94].

The other genetic operator usually used is mutation. It is used to produce the variation of genomes. Mutation is applied stochastically to a child after recombination. It alters one or more genes with a small probability allowing a small amount of random search. As a result of that no point in the whole search space has a zero probability of being visited by the genetic algorithm. Therefore, a mutation operator is used to enhance the diversity and provide a chance to escape from local optima. Many mutation operators were proposed such as inverse, insert, displace, swap, hybrid mutation [82], and heuristic mutation. The first five listed here are realized by small modification of genes. Heuristic mutation was proposed by Cheng and Gen [20], this operator adopts a neighborhood strategy to improve the solution.

Overall the Genetic Algorithms have shown themselves to be useful and efficient when the search space is large, complex or poorly understood. A lot of progress was made recently. In 2006, Carter and Ragsdale propose a new GA chromosome and to them related operators for the Multiple TSP [19]. In 2007, Nguyen described a hybrid GA based on a parallel implementation of a multi population steady-state GA involving local search heuristics [109].

Edge Map Crossover

Edge Map Crossover (EMX) [51] is an implementation of the recombination op-

erator for Genetic Algorithms. It makes use of a so called edge map. Edge map is a table in which each location is placed. For each location there is a list in which the neighbouring locations are listed with this location. Recombination is then established as follows. Choose the first location of one of both parents to be the current location. Second step is to remove the current location from the edge map lists. If the current location still has remaining edges, go to the previous step, otherwise go to the next step. Choose the new current location from the edge map lists of the current location as the one with the shortest edge map list. If there are remaining locations, choose the one with the shortest edge map list to be the current location and return to second step.

```
Example:
Parents: 1-2-3-4-5-6; 2-4-3-1-5-6
Edge map: 1) 2 6 3 5;  2) 1 3 4 6;  3) 2 4 1;
          4) 3 5 2;    5) 4 6 1;    6) 1 5 2 6
1. Random choice: 2,
2. Next candidates: 1 3 4 6, choose from 3 4 6 same#edges, choose 3,
3. Next candidates: 1 4 (edge list 4 < edge list 1), choose 4,
4. Next candidate:  5, choose 5,
5. Next candidate:  1 6 (tie breaking) choose 1,
6. Next candidate;  6, choose 6.
Offspring: 2-3-4-5-1-6
```

Distance Preserving Crossover

Distance Preserving Crossover (DPX) is another implementation of the recombination operator for Genetic Algorithms. It attempts to create a new tour with the same distance to both parents [70]. In order to establish this, the content of the first parent is copied to the offspring and all edges that do not occur in the second parent are removed. The resulting fragments are reconnected without making use of non-overlapping edges of the parents. If edge (i, j) has been destroyed, the nearest available neighbor k of i from the remaining fragments, is selected and the edge (i, k) is added to the tour.

```
Example:  Parents: 5-3-9-1-2-8-0-6-7-4; 1-2-5-3-9-4-8-6-0-7
Fragments: 5-3-9|1-2|8|0-6|7|4
Offspring: 6-0-5-3-9-8-7-2-1-4
```

We procede to next subsection which will in more details describe the hybrid gentic algorithms.

Hybrid Genetic Algorithms - Memetic Algorithms

Hybrid Genetic Algorithms are evolutionary algorithms that include a stage of individual optimization or learning as part of their search strategy. Some, but not

all, of the references can be traced to [48, 68, 133].The most basic Hybrid Genetic Algorithm can be seen below in Algorithm 8 :

Algorithm 8 Hybrid Genetic Algorithm

```
1: procedure HybridGA
2:     Randomly generate an initial population P(t)
3:     while (Termination Criterion Unfulfilled) do
4:         Compute the fitness f(p) ∀p ∈ P(t)
5:         According to f(p) choose a subset of P(T), store them in M(t)
6:         Recombine and mutate individuals in M(t), store results in M'(t)
7:         Improve by local search (M'(t))
8:         Generate P(t + 1) by selecting some individuals from P(t) and M'(t)
9:         t = t + 1
10:    end while
11:    Return best p ∈ P(t − 1)
12: end procedure
```

In the pseudocode of the hybrid genetic algorithm the difference feature from a standard GA is the use of local search. It is used to improve the newly created individuals. The reader should note that this is just one possible way to hybridize a GA with local search. Even though it seems from the Algorithm 8, to be a naive minor change, is in fact a crucial deviation from a canonical GA. Hybrid genetic algorithms are inspired by model of adjustment in natural systems that combine evolutionary adjustment of populations of individuals with individual learning.

In the literature, Hybrid Genetic Algorithms have also been named Memetic Algorithms (MA) [62, 87–89, 104, 113], Genetic Local Searchers (GLS) [102], Lamarckian Genetic Algorithms [105], Baldwinian Genetic Algorithms [90]. The Memetic Algorithms differ from other hybrid evolutionary techniques in that all individuals in the population are local optimum, since after each mutation or recombination, a local search is applied. The name genetic local search (GLS) was firstly used by Ulder in [131] to describe an evolutionary algorithm with recombination and then applied local search. In [17], Bui suggest a GLS algorithm with Lin-Kernighan heuristic as the neighbourhood procedure. They developed a k-point recombination operator with an additional repair mechanism for producing the feasible offspring. In [81] Katayama suggest an evolutionary algorithm with Lin-Kernighan algorithm and small populations, (he used just two individuals) and a heuristic recombination scheme. This approach is likewise to the iterated Lin-Kernighan heuristic but increased diversification is achieved by the crossover of the current solution and the best solution found.

In Chapter 3 we will systematically study approaches to Hybrid Genetic Algorithm. Furthermore, we are going to investigate some of the constructive and local search approaches we have described in this chapter and to compare their performance to our grafted genetic heuristic approache for symmetric Traveling Salesman Problem.

2.2.5 Finding exact solutions for the TSP

Finding the exact solution to a Traveling Salesman Problem with n cities involves to check $(n-1)!$ of possible tours. Evaluation of all possible tours is infeasible for even small instances of TSP. For finding the optimal tour Held and Karp [69] introduced the following dynamic programming formulation: Given a subset of city pointers, discarding the first city, $S \subset \{2, 3, ..., n\}$ and $l \in S$, let $d^*(S, l)$ stand for the length of the shortest path from city 1 to city l, visiting all cities in S in between. For $S = \{l\}, d^*(S, l)$ is defined as d_{1l}. Then the shortest path for larger sets with $|S| > 1$ is:

$$d^*(S, l) = \min_{m \in S \setminus \{l\}} (d^*(S \setminus \{l\}, m) + d_{ml}). \tag{2.9}$$

In conclusion, the minimal tour length for a complete tour which includes returning to city 1 is:

$$d^{**} = \min_{l \in \{2,3,...,n\}} (d^*(\{2, 3, ..., n\}, l) + d_{l1}). \tag{2.10}$$

Using the Equation 2.9 and the Equation 2.10, the quantities $d^*(S, l)$ can be calculated recursively and the minimal tour length d^{**} can be obtained. In a next step, the optimal permutation $\pi = \{1, i_2, i_3, ..., i_n\}$ of city pointers 1 through n can be calculated oppositely, starting with i_n and working consecutively back to i_2. The step take advantage of the fact that a permutation π can be optimal only if

$$d^{**} = d^*(\{2, 3, ..., n\}, i_n) + d_{i_n 1} \tag{2.11}$$

and, for $2 \leq p \leq n - 1$,

$$d^*(\{i_2, i_3, ..., i_p, i_{p+1}\}, i_{p+1}) = d^*(\{i_2, i_3, ..., i_p\}, i_p) + d_{i_p i_{p+1}} \tag{2.12}$$

The complexity of space for storing the values for all $d^*(S, l)$ is $(n-1)2^{n-2}$ which strictly restricts the dynamic programming algorithm to TSP of small sizes. In spite of that for very small TSP instances this approach is fast and efficient [66].

A quite different method can deal with larger instances by using a relaxation of the LP problem. This procedure iteratively tightens the relaxation till a solution is found. This method for solving LP problems is called *cutting plane method* and was introduced by Dantzig, Fulkerson, and Johnson in 1954 [31].

Each iteration of a method begins with using the relaxation $Ax \leq b$, where the polyhedron P determined by the relaxation contains S and is bounded. In addition the optimal solution x^* of the relaxed problem can be reached using standard LP solvers. If the x^* found belongs to S, the optimal solution of the original problem is obtained. If not then a linear inequality can be found which is content by all points in S but violated by x^*. This inequality is called a cutting plane or cut. If no additional cutting planes can be found or the improvement becomes very small, the problem is branched into two subproblems. These sub-problems can be minimized individually. Branching is done iteratively that leads to a binary tree of subproblems. Then each of a subproblem is solved without further branching or is found to be

irrelevant. Irrelevant means that relaxed version previously produces a longer path than a solution of another subproblem. This method is called branch and cut and was introduced by Padberg and Rinaldi, in 1990 [116]. The branch and cut method is a variation of the branch and bound procedure presented by Land and Doig, in 1960 [91].

The initial polyhedron P used by Dantzig [31] contains all vectors x such that for all $e \in E$ is $0 \leq x_e \leq 1$. Furthermore, in the resulting tour each city is connected to exactly two other cities. Different methods for finding cuts to prevent sub-tours (*sub-tour elimination inequalities*) and to ensure an integer solution *Gomory cuts*, were developed over time [66]. At present the most effective implementation of this method is *Concorde* described in [4]. Concorde is a computer code for the symmetric traveling salesman problem. The code is written in the *AnsiC* programming language. At the time of writing this dissertation the Concorde's TSP solver has been used to obtain the optimal solutions to 106 of the 110 instances from TSPLIB [122]. In Chapter 3 the Concorde's TSP solver was used for computing the lower bound for the quality of solutions for tested algorithms. Furthermore, in Chapter 4 it was used as a solver for the Traveling Salesman Problem.

Chapter 3

Grafted Genetic Algorithm for Traveling Salesman Problem

Results of this chapter are published in the following articles:

- M. Djordjevic, Influence of Grafting a Hybrid Searcher Into the Evolutionary Algorithm, *Proceedings of the Seventeenth International Electrotechnical and Computer Science Conference, ERK 2008.*, Portoroz, (2008), 115–118.
- M. Djordjevic, and M. Tuba, and B. Djordjevic, Impact of Grafting a 2-opt Algorithm Based Local Searcher Into the Genetic Algorithm, *Proceedings of the 9th WSEAS International Conference on Applied Informatics and Communications, AIC 2009.*, Moscow, (2009), 485–490.
- M. Djordjevic, and A. Brodnik, Quantitative Analysis of Separate and Combined Performance of Local Searcher and Genetic Algorithm, *Book of Abstracts of International Conference on Operations Research, OR 2011.*, Zurich, (2011), 130.
- M. Djordjevic, and A. Brodnik, Quantitative Analysis of Separate and Combined Performance of Local Searcher and Genetic Algorithm, *Proceedings of the 33rd International Conference on Information Technology Interfaces, ITI 2011.*, Dubrovnik, (2011), 515–520.

3.1 Introduction

Genetic Algorithms (GA), which was in detail described in Section 2.2.4 use some mechanisms inspired by biological evolution [71]. They are applied on a finite set of individuals called population. Each individual in a population represents one of the feasible solutions of the search space. Mapping between genetic codes and the search space is called encoding and can be binary or over some alphabet of higher cardinality. Good choice of encoding is a basic condition for successful application of a genetic algorithm. Each individual in the population is assigned a value called fitness. Fitness represents a relative indicator of quality of an individual compared to other individuals in the population. Selection operator chooses individuals from the

current population and takes the ones that are transferred to the next generation. Thereby, individuals with better fitness are more likely to survive in the population's next generation. The recombination operator combines parts of genetic code of the individuals (parents) into codes of new individuals (offsprings). Such a mixing of genetic material enables that well-fitted individuals or their relatively good genes give even better offspring. By a successive application of selection and crossover, the diversity of genetic material can be decreased which leads to a premature convergence in a local optimum which may be far from a global one.

The components of the genetic algorithm software system are: Genotype, Fitness function, Recombinator, Selector, Mater, Replacer, Terminator, and in our system a (Local) Optimizer which is a new extended component. In this chapter we study a well defined problem of a Traveling Salesman Problem (TSP), details about this problem can be found in Section 2.1. In the TSP a set $\{C_1, C_2, ...C_N\}$ of cities is considered and for each pair $\{C_i, C_j\}$ of distinct cities a distance $d(C_i, C_j)$ is given. The goal is to find an ordering π of the cities that minimizes the quantity

$$\sum_{i=1}^{N-1} d(C_{\pi(i)}, C_{\pi(i+1)}) + d(C_{\pi(N)}, C_{\pi(1)}). \tag{3.1}$$

This quantity is referred to as the minimum tour length since it is the length of the tour a salesman would make when visiting the cities in the order specified by the permutation, returning at the end to the initial city. We will concentrate in this paper on the symmetric TSP in which the distances satisfy $d(C_i, C_j) = d(C_j, C_i)$ for $1 \leq i, j \leq N$ and more specificaly to the Euclidean distance. The TSP is known to be *NP-hard* [54], even under substantial restrictions. The case with Euclidean distance is well researched and there are algorithms which perform well even on very large cases [4].

The 2-opt is a simple greedy optimization algorithm for the TSP, it is in details described in Section 2.2.3. The main idea behind it is to take a route that crosses itself and reorder it so that it does not cross itself any more reducing the tour length. An exchange step consists of removing two edges from the current tour and reconnecting the resulting two paths in the best possible way, as shown in Figure 2.1. Once we choose the two edges to delete, we do not have a choice about which edges to add there is only one way to add new edges that results in a valid tour.

The 2-opt optimization will be used to hybridize GA meta-heuristic to solve TSP. Although the 2-opt algorithm [45, 70] performs well and can be applied to TSP with many cities, it finds only a local minimum. The nearest neighbour algorithm, details can be found in Section 2.2.2, is one of the most intuitive heuristic algorithms for the TSP. It is a greedy method for solving the TSP. The genetic algorithm considered in this paper are hybrid genetic algorithms, details are in Section 2.2.4, incorporating local optimization (cf. Memetic Algorithms, [104]). One example of hybridization of genetic algorithms is shown in [125].

3.2 Grafted GA for the TSP

Grafting in botany is when the tissues of one plant are affixed to the tissues of another. Grafting can reduce the time to flowering, shorten the breeding program, etc. Similarly we introduced into a canonical GA a local optimizer - we grafted GA or we hybridized it. This way we (locally) optimize each genome in an evolution process. There exist a number of local optimizers, which can be used on their own as a greedy solution to NP-hard problems - e.g. Freisleben in [51] used a k-opt heuristics. There exists even its hardware implementation [72].

The pseudo-code of our Grafted Genetic Algorithm (GGA) is listed in Algorithm 9.

Algorithm 9 Grafted Genetic Algorithm

```
1:  procedure GGA
2:      t = 0
3:      p(t) := Initialize()
4:      q(t) := Evaluate(P(t))
5:      while (q(t) < q_expected) and (t < t_max) do
6:          sel := Select(P(t))
7:          mat := Mate(sel)
8:          rec := for each pair m ∈ mat do Recombine(m)
9:          loc := for each genome r ∈ rec do Optimize(r)
10:         P(t + 1) := Replace(loc, P(t))
11:         q(t + 1) := Evaluate(P(t + 1))
12:         evaluate (P(t + 1))
13:         t := t + 1
14:     end while
15: end procedure
```

As usual, our algorithm stops either when the expected quality of solution is reached or when the maximum number of generations t_{max} is passed. The former can be measured in various terms starting from the absolute quality value to the relative diversification of population (e.g. standard deviation). On the other hand the later is measured either in the number of generations or in the total time elapsed. Tournament Selector (line 6 in the algorithm) places groups of genomes from the population together, creating the groups from top to bottom with respect to the enumerative ordering of the genomes in the population and selects the best of the genomes within this group. This is repeated until the required amount of genomes is selected. The Random Mater (line 7 in the algorithm) is a simple way of mating parents. It mates the parents as enumerated in the population at random using the mating size to create groups until no more groups can be created. The new offspring only replacer is the implementation of the classical replacement strategy that simply only allows the offspring to survive. Thus the genomes from the next generation replace the entire current population.

We studied two versions of recombination (line 8 in the algorithm). The first, *Edge map crossover, emx* for short [104], uses a so-called edge map $EM[^*]$ that

contains a list of neighbouring cities for each city. First the operator for each edge (u, v) in a parent genomes A and B, adds v to the edge-map list $EM[u]$ and u to $EM[v]$ for a symmetric version of TSP. Then the operator works as follows:

1. Pick a random city to be the current location u.
2. Remove the current location u from all edge map lists $EM[^*]$.
3. If the current location $EM[u]$ still has remaining edges, go to step 4, otherwise go to step 5. item Choose the new current location u' from the edge map list $EM[u]$ as the one with the shortest edge map list $EM[u']$. Set $u := u'$ and go to step 2.
4. If there are left any locations, choose as the new current location u' the one with the shortest edge map list $EM[u']$. Set $u := u'$ and go to step 2.

The edge map crossover is in more details presented in Section 2.2.4. The second recombination operator we studied was a *distance preserving crossover, dpx* for short [51]. It creates a new tour (offspring) preserving the same distance in the number of edges to both parents. In detail, the operator dpx creates an offspring C from parents A and B as follows:

1. Pick at random one of the parents ($wlog.A$) and copy it to C.
2. If $(u, v) \in C$ and $(u, v) \notin B$ delete it from C. At this point C contains fragments of connected cities $(u_l \to v_l), \ldots, (u_k \to v_k)$, where in some fragments $u_i = v_i$. We call set of u_i and v_i the end-points of C, EP_c.
3. Pick at random a city x from EP_c and delete it from EP_c.
4. Pick from EP_c the closest city y to x so that there is edge (x, y) neither in A nor in B. Delete y from EP_c.
5. Merge fragments $(u_i \to y)$ and $(x \to v_j)$ into $(u_i \to v_j)$.
6. If EP_c is not empty, go to step 3.

The distance preserving crossover is in more details described in Section 2.2.4. Although both recombination operators produced offsprings from valid parents, they produced them in a random way. Because of randomization, the selection and mating (lines 5 and 6 respectively) lost their role. Moreover, the randomization is a very good source of diversification. However, we are missing in our meta-heuristic the process of specialization.

This was the reason to extend our algorithm with a specialization step - we *grafted* a canonical genetic algorithm with a local optimizer and obtained a *grafted genetic algorithm, GGA* [35], [38], [36], [37]. We did not use a k-opt heuristics due to its complexity, but rather its simpler version 2-opt explained in the previous section (see Figure 2.1). The hybridization occurs in line 9 of the pseudocode of our algorithm given above.

3.3 Experiment

For testing our strategy and comparing it to other solutions we used the instances of symmetric traveling salesman problem found on TSPLIB [122]. We used relatively small instances, for which best solutions are known. The goal of this research was not to find a better algorithm, but rather to study on a controlled environment the impact of grafting a genetic algorithm.

In the first experiment we used 20 instances, with different sizes in a range from 14 to 150 cities per instance (look in Table 3.1). We studied our method (grafted genetic algorithm (GGA)) using two different recombination operators: an edge map crossover (GGAemx) and a distance preserving crossover (GGAdpx). As the upper and lower limits on the quality of solution we used greedy heuristic and Concorde [4] respectively. For the sake of completeness we compared our method also with 2-opt heuristic itself and with a canonical genetic algorithm. The main difference between our method and canonical genetic algorithm is that we use local optimizer in every generation of the algorithm.

In the second experiment we studied what happens if we do not use local optimization in all generations – in test we used it in 10, 20, 30, 40, 50, 60, 70, 80 and 90 percents of the generations. Furthermore, for each percentage we applied local optimization in three different ways: at random generations, at the initial generations and at the ending ones.

All experiments were conducted on a computer with Pentium(R) 2.8 GHz CPU and Windows 7 operating system. In our results we cannot directly compare the running times of different solutions as they were implemented in different programming languages. On one hand we used as a development environment for GGA the Java written *EA Visualizer* [15], while *Concorde* is an *AnsiC* application. However, we can compare running times of GGA for different instances and cases explained before.

3.4 Results

We present results separately for the first and for the second experiment. In both experiments we used instances of TSP from a *TSPLIB* of various sizes. The name of the instance also contains its size (the number of sites, cf. the first columns of Table 3.1). Next, in the experiments we measured three quantities: the wall clock time, the number of generations and the quality of the result. The later was measured against the optimal solution obtained by Concorde. The quality of algorithm A is defined as

$$q_A = \frac{l_A - l_C}{l_C} \tag{3.2}$$

where l_C is a path length obtained by Concorde and l_A is a path length obtained by A. We express the quality always in percents, where, for example, 4% means 4% worse than Concorde.

Let us look first at the results of the first experiment (cf. Table 3.1), in which we compared our GGA against greedy algorithm and Concorde. We also compared it against canonical genetic algorithm (GA). The termination condition in all genetic

Table 3.1: Five techniques for solving Euclidean TSP: greedy, 2-opt heuristic, GA with edge map crossover, GA with distance preserving crossover, grafted versions of the later and Concorde.

Name	**Greedy**	**2-opt**	**GAemc**			**GAdpc**			**GGAemc**			**GGAdpc**			**Concorde**	
	quality	*quality*	*quality*	*gen.*	*time*	*quality*	*gen.*	*time*	*qual.*	*gen.*	*time*	*qual.*	*gen.*	*time*	*opt*	*time*
burma14	8.32%	5.71%	0%	74	3.4	0%	81	3.5	0%	7	0.6	0%	6	0.5	3323	0.1
ulysses16	10.42%	7.15%	0%	136	4.1	0%	125	4.4	0%	9	0.7	0%	9	0.7	6859	0.2
ulysses22	12.54%	7.87%	0%	1267	14.7	0%	1328	16.4	0%	8	0.6	0%	8	0.7	7013	0.2
bayg29	13.37%	6.38%	0%	1345	19.4	0%	1137	17.6	0%	13	1.3	0%	14	1.4	1610	0.3
bays29	12.87%	5.37%	0%	2185	29.2	0%	2643	34.1	0%	12	1.2	0%	12	1.2	2020	0.3
dantzig42	14.06%	7.11%	0%	4704	79.8	0%	4232	74.6	0%	10	1.3	0%	9	1.3	699	0.5
att48	13.98%	8.47%	0%	4807	85.2	0%	5213	91.3	0%	22	2.2	0%	23	2.3	33522	0.6
eil51	15.24%	7.67%	4.21%	5482	100.0+	5.23%	5489	100.0+	0%	33	3.9	0%	30	3.8	426	0.3
berlin52	14.82%	7.45%	0%	2037	33.7	4.92%	5021	100.0+	0%	15	1.7	0%	15	1.7	7542	0.4
st70	13.17%	7.84%	5.12%	5259	100.0+	5.72%	5198	100.0+	0%	20	4.1	0%	19	4.1	675	0.5
eil76	14.47%	8.15%	6.56%	5347	100.0+	7.24%	5298	100.0+	0%	53	4.5	0.19%	49	4.4	538	1.3
pr76	13.96%	9.95%	4.18%	5218	100.0+	5.36%	5191	100.0+	0%	42	4.1	0%	43	4.2	108159	1.2
gr96	16.32%	7.14%	4.98%	5191	100.0+	5.71%	5090	100.0+	0%	73	8.4	0.13%	73	8.4	55209	1.6
rat99	14.79%	7.41%	5.31%	5114	100.0+	7.12%	5011	100.0+	0%	74	11.9	0.17%	70	11.7	1211	1.7
kroA100	12.37%	8.07%	5.12%	5072	100.0+	6.58%	4971	100.0+	0%	24	3.6	0.18%	22	3.5	21282	1.7
kroB100	16.58%	7.19%	6.14%	5041	100.0+	5.92%	4816	100.0+	0%	39	5.8	0.21%	36	5.7	22141	1.7
kroC100	10.47%	11.19%	4.87%	5121	100.0+	6.78%	4923	100.0+	0.10%	34	5.3	0.19%	28	5.1	20749	1.8
kroD100	14.81%	7.74%	5.07%	4976	100.0+	8.12%	4951	100.0+	0%	31	5.6	0.29%	25	5.3	21294	1.5
lin105	16.60%	9.85%	6.72%	4756	100.0+	6.51%	4803	100.0+	0.01%	26	4.6	0.17%	25	4.6	14379	1.3
ch150	19.62%	11.72%	7.22%	4512	100.0+	8.77%	4460	100.0+	0.22%	88	15.2	0.32%	86	15.1	6528	7

algorithms was: either standard deviation of genomes was 0 ((local) minimum was reached) or time 100 seconds time limit expired.

We first look at the quality of results, than at the running time and finally comment on a trade-off between the quality and the running time. The last column of Table 3.1 gives results of Concorde and actual path length in *opt* column. On the other side of the table is a greedy approach, which quality (column *quality* computed using Equation 3.2) is mostly in the range between 10% and 20%. However, application of a simple 2-opt heuristic on a randomly generated tour improves the quality to approximately 10% or even better. On the other hand use of GA further improves the quality to approximately 5%. Note, that runs in cases with 70 or more sites terminated due to time limit and hence minimum was not reached. All these results were expected.

The long running time of GA was a reason to graft (or hybridize) the GA with local optimization. The result was substantial decrease in running time. In all cases for GGAemx and for GGdpx the runs were terminated upon reaching the minimum. The reached minimum was, however, the local one. Nonetheless, we showed, that the combination of two methods improved the quality of results in a synergy. The quality of result was bellow 1% off the optimum.

The sixth column in Table 3.1 describes results obtained by GGAemx. In 17 out of 20 considered cases an optimal solution was found. Remaining three instances differ from optimal solution in 0.01, 0.10 and 0.22 percent. The solutions were found in relatively few generations and very fast. Execution times were 0.6 to 15.2 seconds.

The seventh column in Table 3.1 corresponds to results of GGAdpc. In 11 out of 20 considered cases an optimal solution was found. In remained 9 cases, delivered solutions differ from optimal in range from 0.13 to 0.32 percent. The running time and number of generations of GGAdpx, in comparison with GGAemx, are slightly lesser, particularly in the lowermost part of the table which represents more complex instances.

Quantitative results on test cases from *TSPLIB* show that grafted algorithms, GGAemx and GGAdpx, have advantages. Even when their's components have serious drawbacks, their grafted combinations exhibits a very good behaviour. Results on examples from *TSPLIB* show that this grafted method combines good qualities from both methods applied and significantly outperforms each individual method.

The running times in Table 3.1 are given for all algorithms but greedy and simple 2-opt heuristics. The later ones had running time in the range between half a second and a second and a half. However, since all algorithms but Concorde were programmed in Java, their running times are not directly comparable. Nonetheless, the relative increase in time as function of a problem size can be compared, and this shows us approximately 25 times increase for GGAemx, 30 times increase for GGAdpx and even 70 times increase for Concorde. Note also, that GGAemx and GGAdpx performed approximately the same, which shows that the recombination operator has no major influence on a final result.

In the second experiment we studied the influence of grafting (hybridization) on running time and quality of solution. In this experiment we used only grafted GA with edge map crossover (GGAemx) and only eleven cases from a TSPLIB (cf. Table 3.2). In the experiment we were increasing the number of generations in which

Table 3.2: Partial grafting of a genetic algorithm

Name	GAemx			10				20				30				40				50			
				rnd	begin	end	f.a	rnd	begin	end	f.a	rnd	begin	end	f.a	rnd	begin	end	f.a	rnd	begin	end	f.a
	q		*t*	*q*	*q*	*q*	*t*	*q*	*q*	*q*	*t*	*q*	*q*	*q*	*t*	*q*	*q*	*q*	*t*	*q*	*q*	*q*	*t*
eil76	8.93%		0.8	2.46%	2.13%	1.25%	1.2	1.80%	0.99%	0.96%	1.5	1.62%	0.92%	0.77%	1.8	1.58%	0.44%	0.22%	2.2	1.14%	0.37%	0.18%	2.6
pr76	5.39%		0.9	0.60%	0.41%	0.34%	1.3	0.32%	0.24%	0.19%	1.7	0.25%	0.17%	0.10%	2.1	0.20%	0.16%	0.09%	2.4	0.17%	0.11%	0.07%	2.7
gr96	6.46%		1.7	1.68%	0.78%	0.70%	2.3	1.37%	0.59%	0.55%	2.9	0.94%	0.59%	0.55%	3.6	0.78%	0.59%	0.55%	4.2	0.82%	0.55%	0.47%	4.9
rat99	6.14%		1.9	2.53%	1.82%	1.62%	2.6	2.67%	0.90%	0.71%	3.3	2.81%	0.62%	0.56%	4.1	2.13%	0.53%	0.39%	5.2	1.28%	0.43%	0.38%	6.3
kroA100	6.67%		0.6	1.09%	0.73%	0.38%	0.9	0.84%	0.38%	0.33%	1.2	0.19%	0.22%	0.12%	1.5	0.18%	0.08%	0.03%	1.8	0.16%	0.03%	0.02%	2.1
kroB100	7.02%		0.8	1.61%	1.15%	1.02%	1.3	1.26%	0.70%	0.53%	1.8	0.72%	0.70%	0.42%	2.3	0.71%	0.47%	0.35%	2.8	0.76%	0.40%	0.38%	3.3
kroC100	6.61%		0.7	2.20%	1.05%	0.99%	1.2	1.19%	0.90%	0.75%	1.6	0.97%	0.63%	0.52%	2.1	0.79%	0.44%	0.37%	2.5	0.74%	0.38%	0.36%	2.9
kroD100	7.67%		0.8	2.20%	1.87%	2.11%	1.3	2.39%	2.02%	1.47%	1.8	1.44%	1.17%	0.97%	2.3	1.26%	0.89%	0.67%	2.8	0.97%	0.54%	0.46%	3.3
lin105	8.54%		0.5	1.50%	1.11%	1.19%	0.9	0.89%	0.70%	0.50%	1.4	0.83%	0.40%	0.41%	1.8	0.71%	0.37%	0.29%	2.2	0.46%	0.23%	0.23%	2.6
ch150	8.69%		5.4	2.94%	2.52%	2.34%	6.2	2.17%	1.89%	1.83%	6.9	1.77%	1.58%	1.37%	7.8	1.63%	1.46%	1.36%	8.7	1.31%	1.19%	0.92%	9.6
pr439	10.45%		3.7	4.92%	4.35%	3.48%	11	4.59%	3.43%	2.96%	18	4.04%	3.16%	2.81%	25	3.34%	2.92%	2.56%	36.8	3.62%	3.13%	2.45%	45
	GGAemx			**90**				**80**				**70**				**60**							
				rnd	begin	end	f.a	rnd	begin	end	f.a	rnd	begin	end	f.a	rnd	begin	end	f.a				
	q		*t*	*q*	*q*	*q*	*t*	*q*	*q*	*q*	*t*	*q*	*q*	*q*	*t*	*q*	*q*	*q*	*t*				
eil76	0.04%		4.5	0.15%	0.04%	0.04%	4.1	0.18%	0.07%	0.04%	3.7	0.44%	0.15%	0.11%	3.3	1.07%	0.22%	0.11%	2.9				
pr76	0.04%		4.1	0.07%	0.04%	0.04%	3.8	0.12%	0.10%	0.05%	3.5	0.11%	0.10%	0.06%	3.2	0.15%	0.11%	0.06%	2.9				
gr96	0.12%		8.4	0.23%	0.16%	0.12%	7.7	0.27%	0.23%	0.20%	7	0.39%	0.35%	0.27%	6.3	0.74%	0.35%	0.31%	5.6				
rat99	0.00%		11.9	0.07%	0.00%	0.00%	10.8	0.56%	0.05%	0.02%	9.7	0.61%	0.16%	0.05%	8.5	1.13%	0.30%	0.25%	7.4				
kroA100	0.00%		3.6	0.00%	0.00%	0.00%	3.3	0.00%	0.00%	0.00%	3	0.01%	0.00%	0.00%	2.7	0.05%	0.00%	0.00%	2.4				
kroB100	0.10%		5.8	0.22%	0.12%	0.09%	5.3	0.31%	0.37%	0.22%	4.9	0.52%	0.20%	0.24%	4.3	0.81%	0.30%	0.29%	3.7				
kroC100	0.23%		5.3	0.37%	0.32%	0.23%	4.8	0.57%	0.56%	0.22%	4.3	0.52%	0.34%	0.29%	3.7	0.55%	0.32%	0.32%	3.3				
kroD100	0.08%		5.6	0.32%	0.28%	0.08%	5.2	0.58%	0.31%	0.26%	4.7	0.61%	0.45%	0.40%	4.3	0.88%	0.53%	0.45%	3.8				
lin105	0.10%		4.6	0.12%	0.09%	0.10%	4.2	0.11%	0.15%	0.10%	3.8	0.13%	0.12%	0.09%	3.4	0.21%	0.17%	0.12%	3.0				
ch150	0.30%		15.2	0.81%	0.35%	0.30%	14.1	0.86%	0.42%	0.37%	13	0.96%	0.69%	0.54%	12	1.16%	0.97%	0.84%	10.7				
pr439	1.30%		91.8	1.72%	1.66%	1.34%	78.9	2.28%	2.14%	1.97%	70	2.78%	2.18%	2.03%	62	3.26%	2.91%	2.31%	52.4				

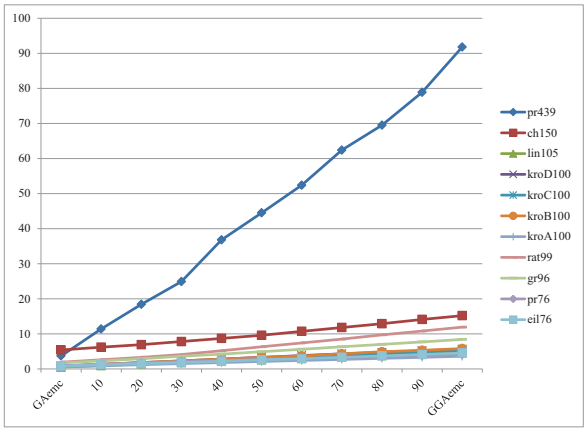

Figure 3.1: The running time as a function of amount of hybridization.

we applied hybridization by 10 percents: from 0% – column *GAemx* in Table 3.2 till 100% – column *GGAemx*. Moreover, we also varied the generations in which we applied the hybridization: either in random generations (column *rnd*), in the beginning ones (*begin*) or in the ending ones (*end*). The column *f.a* gives the running time, while the numbers in columns *q* give the quality computed using Equation 3.2.

We first observe, that application of grafting in the last generations gives the best results. This is reasonable, as in general in this phase of meta-heuristics we apply mostly intensification and not that much more diversification. On the other hand it is interesting that the worse results were obtained when grafting was applied in random generations. Nonetheless, since in practice the algorithm does not know which are the last generations, we would need to simulate the behaviour. There are two possibilities how to do it: either, when time limit is reached run the algorithm for some more runs and apply hybridization or apply hybridization more and more frequently as the number of generations increases. The later approach is also in line with other meta-heuristics like simulated annealing.

The running time obviously linearly increases as we increase the amount of hybridization (see Figure 3.1). Similarly the quality of solution also improves as we increase the hybridization. In Figure 3.2 we see that for the case *pr439* with 439 sites the quality of solution improved from over 10% at no hybridization to approximately 3% at half hybridization and to 1.3% off the optimal at total hybridization. Similar behaviour can be observed at other cases (cf. Figure 3.3). Note, that even small hybridization of 10% drastically improves solution – e.g. for the *pr439* case to only about 4% off the optimal. On the other hand, further hybridization keeps improving result and it is up to the user to decide how much hybridization does she want to employ.

Probably the decision on the amount of hybridization should be made considering the running time. As seen in Figure 3.1 the number of sites increases the steepness of the function. Therefore high hybridization at large cases would probably increase the running time substantially. However, from our experiments seems to follow that also lower amounts of hybridization give satisfactory results (see Figure 3.3).

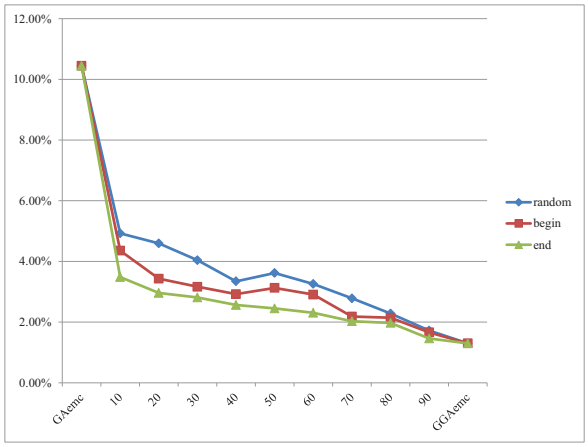

Figure 3.2: Quality results for case *pr439* as the amount of hybridization increases.

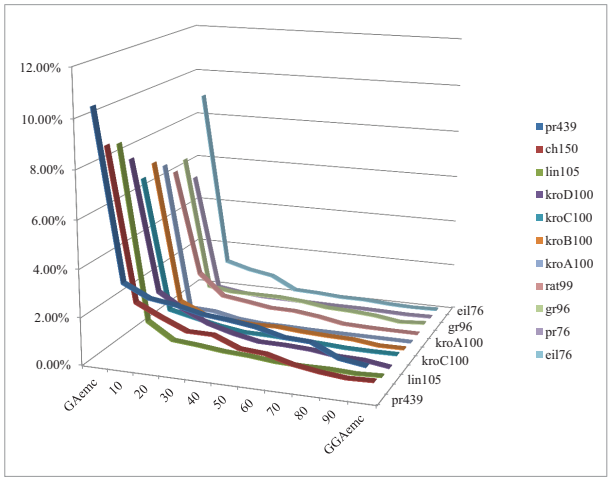

Figure 3.3: Quality results for all cases using end-hybridization.

3.5 Conclusions

The goal of this chapter was to investigate influence of grafting a 2-opt based local searcher into the canonical genetic algorithm, for solving the Traveling Salesman Problem. It is known that genetic algorithms are very successful when implemented for many NP-hard problems. However, they are much more effective if some specific knowledge about particular problem is utilized. In our first experiment we compared two direct techniques, with our grafted genetic algorithms. Solutions from Concorde and greedy algorithm were added for better comparison. Quantitative results on test cases from *TSPLIB* show that grafted algorithms have advantages. Even when both components have serious drawbacks, their grafted combinations exhibits a very good behaviour. Results on examples from *TSPLIB* show that this method combines good qualities from both methods applied and significantly outperforms each individual method.

Our experiments further show that the best results are obtained when hybridization occurs in the last generations of the GA. This seems to be in line with classical meta-heuristic algorithms like simulated annealing, which stop their diversification in the last iterations. We showed, that even a small hybridization substantially improves the quality of the result. Moreover, the hybridization in fact does not deteriorate the running time too much.

This chapter opens a number of interesting questions. The first one is related to the size of the problem. Namely, how do our GGA behave in cases, where Concorde can no more compute the optimal solution. Next, can we use some other local optimization technique instead of 2-opt and how would then behave our GGA. A very interesting question would also be how would our technique work in other NP-hard problems. For example in 3-SAT or CLIQUE.

Chapter 4

Traveling Visitor Problem

Results of this chapter are published in the following articles:

- M. Djordjevic, A. Brodnik and M. Grgurovic, The Traveling Visitor Problem and Algorithms for Solving It, *Book of Abstracts of 3rd Student Conference on Operational Research, SCOR 2012.*, Nottingham, (2012), 26.
- M. Djordjevic, A. Brodnik and M. Grgurovic, The Traveling Visitor Problem and the Koper Algorithm for Solving It, *Book of Abstracts of 25th Conference of European Chapter on Combinatorial Optimization, ECCO 2012.*, Antalya, (2012), 10.

4.1 Introduction

In the Traveling Salesman Problem (TSP) a set $\{C_1, C_2, ...C_N\}$ of cities is considered and for each pair (C_i, C_j) where $i \neq j$, a distance $d(C_i, C_j)$ is given. The goal is to find a permutation π of the cities that minimizes the quantity

$$\sum_{i=1}^{N-1} d(C_{\pi(i)}, C_{\pi(i+1)}) + d(C_{\pi(N)}, C_{\pi(1)}). \tag{4.1}$$

This quantity is referred to as the tour length since it is the length of the tour a salesman would have to travel when visiting the cities in the order specified by the permutation π, returning at the end to the initial city. We will concentrate in this chapter on the symmetric TSP (STSP) in which the distances satisfy $d(C_i, C_j) = d(C_j, C_i)$ for $1 \leq i, j \leq N$. The TSP is known to be *NP-hard* [76]. The case with symmetric distances has been well studied and there are many algorithms which perform well even on large cases [3, 5]. In the literature the Traveling Salesman Problem is usually represented and considered as a graph theoretical problem, see [63, 75]. For more details about TSP see Section 2.1.

An instance of the STSP can be seen as a complete graph $G = (V, E)$ where the set of vertices V is given by the cities and edges between each city in the graph with corresponding edge weights $d(C_i, C_j)$. The STSP then translates to the problem of finding a Hamiltonian Tour of minimal length in the graph G.

Applications of the TSP and its variations go way beyond the route planning problem of a traveling salesman and span over several areas of knowledge including mathematics, computer science, operations research, genetics, engineering, and electronics. In addition, there are many different variations of TSP which are described and explored in the literature and also variations derived from everyday life. Some of them are: *machine scheduling problems* [8, 63], the*time dependent TSP* [59], the *delivery man problem* which is also known as the *minimum latency problem* and the *traveling repairman problem*, for details on these problems, we refer to Section 2.1.3.

The Traveling Tourist Problem [95] is a problem in which a tourist wishes to see all monuments (nodes) in a city, and so must visit each monument or a neighbour thereof. It is assumed that a monument is visible from any of its neighbours and therefore, the edges represent lines of sight. The resulting walk will therefore visit a subset of all nodes in the graph. The Traveling Tourist Problem shares a similar name with our problem however it is a very different problem.

The STSP can be also solved using the Grafted Genetic Algorithms (GGA) as was shown in [35], [38], [36], [37] and in Section 3. The currently most efficient implementation of the branch-and-cut method which was introduced by Padberg and Rinaldi [116] for solving the symmetric case of TSP is named *Concorde* [4]. Concorde's TSP solver has been used to obtain the optimal solutions to the full set of 110 TSPLIB instances, the largest having 85,900 cities. For more details see Section 2.2.5. Finally, in a graph G besides finding the shortest closed walk we can also find the shortest path between any pair of vertices. This problem is in the literature known as *all-pairs shortest path problem*(APSP) [28]. It aims to compute the shortest path from each vertex u to every other vertex v. The Floyd-Warshall algorithm [28] is an efficient algorithm to find all-pairs shortest paths on a graph G. The all-pairs shortest path problem and the Floyd-Warshall algorithm are in more details described in Sections 2.1.5 and 2.1.5.

4.2 Traveling Visitor Problem

Suppose that visitors have arrived in a hotel in some town, with a desire to visit all interesting sites in a city exactly once and to come back to the hotel at the end of their journey. Visitors in general move from one place to another using the existing streets, walking trails and pedestrian zones. The goal is to minimize the visitors traveling distance.

The Traveling Visitor Problem is a version of the Traveling Salesman Problem with a difference that the traveling visitor, during its visit of sites, can not fly over the buildings in the city, instead visitors must go around these obstacles. This difference is demonstrate in Figure 4.1. This means that the Euclidean distances [57, 113], as we know them in the Euclidean TSP, are in this case not valid (direct edge from i to j in Figure 4.1). Visitors use the walking paths and pedestrian zones of variable length. These limits determine the weight of edges connecting the vertices in the graph.

The Traveling Visitor Problem is stated as: given a connected, weighted graph $G = (V, E, c)$, with a set of vertices $V = S \cup X$ and $S \cap X = \emptyset$, S is the set of

interesting sites in the city (vertices i and j in Figure 1.1), X is the set including the nodes corresponding to crossroads in the city (vertices k and m in Figure 1.1), a set of edges E, and a cost of traveling c. The goal is to find the shortest closed walk through all vertices from S, according to c in graph G, although we may travel through vertices from X.

The concepts we summarised above can be modified easily to take the directions of the edges into account. The asymmetric traveling visitor problem (ATVP) is then similar to the symmetric TVP above, i.e. it is the problem of finding a closed walk of minimal length in a weighted graph. The Euclidean TVP, or planar TVP, is the TVP with the distance being the ordinary Euclidean distance. The Euclidean TVP is then a particular case of the metric TVP, since distances in a plane obey the Euclidean triangle inequality.

This problem, by the knowledge of the authors, has no references in publications due date of writing it.

4.2.1 Algorithms for solving TVP

One simple approach of solving TVP is to visit all places as are ordered in the city's tourists maps and then come back to the starting site. The result of this method depend directly on the order in which the interesting sites are listed on the map and does not necessary find the shortest closed walk through all tourist sites.

The first proposed method for solving the Traveling Visitor Problem is the Naïve algorithm, shown in Algorithm 10. In the first line of pseudocode we can distinguish next parameters: S is the set of interesting sites in the city, X is the set of crossroads in the city, a set of edges E, and W represents the distance matrix of the graph G, $(S \cup X \times S \cup X)$. The First step of the algorithm is to solve a TSP considering every node in S. Then a tour, denoted by T, is obtained. In the next step, a distance matrix Z is constructed by solving the APSP for every node in S. Note that to calculate these distances we do use distance matrix W (nodes in $S \cup X$). Finally, we calculate the final cost combining the shortest path solution obtained in the second step with the optimal tour obtained in the first step.

Algorithm 10 Naïve Algorithm

```
1: procedure NAÏVE(S,X,E,W)
2:     T ← TSP(S)
3:     Z ← APSP(S ∪ X, E, W)
4:     cost ← 0
5:     for all (i, j) ∈ T : do
6:         cost ←cost +Z_ij
7:     end for
8: end procedure
```

The second proposed method for solving the Traveling Visitor Problem is our Koper algorithm, shown in Algorithm 11. The first line of pseudocode contains the same parameters as Naïve algorithm. In the first step we find all-pairs shortest paths in our graph G. As an input a distance matrix W is used and as the output a distance

matrix Z is obtained ($Z \leftarrow S \times S$). In the next step we solve the Traveling Salesman Problem on the distance matrix Z. Furthermore, we get the solution T, which is a solution for Traveling Visitor Problem. Algorithms presented in this section was named so by the author of dissertation.

Algorithm 11 Koper Algorithm

```
1: procedure KOPER(S,X,E,W)
2:     Z ← APSP(S ∪ X, E, W)
3:     T ← TSP(Z)
4: end procedure
```

4.2.2 Adapted Floyd-Warshall algorithm

The problem stated in the previous section is of finding the shortest paths between each pair of vertices u and v, where $u, v \in S$, in the graph G. This can be cast as a run-of-the-mill all-pairs shortest path problem. Indeed, using the Floyd-Warshall algorithm, we can obtain a solution in time $\Theta(|V|^3)$. However, the nature of our problem is somewhat more restrictive: we are only interested in the shortest paths between $S \times S$, yet we would still like the paths to go through vertices from the set X if they reduce the overall path length. In contrast, the Floyd-Warshall algorithm computes a shortest paths between $V \times V$. To this end, we propose a simple modification which reduces the running time, albeit not asymptotically. The Floyd-Warshall algorithm is shown in Algorithm 12, where W is the distance matrix of the graph G.

Algorithm 12 Floyd-Warshall

```
1: procedure FLOYD-WARSHALL(V,W)
2:     for all k ∈ V do
3:         for all i ∈ V do
4:             for all j ∈ V do
5:                 W_ij := min(W_ij, W_ik + W_kj)
6:             end for
7:         end for
8:     end for
9: end procedure
```

Let $x = |X|$ and $s = |S|$. Using these quantities, the number of iterations of the Floyd-Warshall algorithm can be written as $(s + x)^3 = s^3 + x^3 + 3s^2x + 3x^2s$. We offer a different approach, shown in Algorithm 13.

The number of iterations of algorithm 13 can be plainly seen to equal: $s^3 + x^3 + s^2x + x^2s$. The best gain, when compared to Floyd-Warshall, is when $s = x$ which amounts to exactly one half of all iterations of the Floyd-Warshall algorithm. Although it takes fewer iterations, it also computes fewer shortest paths, since we are only interested in $S \times S$. We will prove the correctness of Algorithm 13 by appealing to the graph shown in Fig. 4.2.2.

In order to examine how Algorithm 13 works, it is helpful to visualize sets of

Algorithm 13 Adapted Floyd-Warshall Algorithm

```
1:  procedure ADAPTED(S,X,W)
2:      FLOYD-WARSHALL(X, W)
3:      for all k ∈ X do
4:          for all i ∈ X do
5:              for all j ∈ S do
6:                  W_ij := min(W_ij, W_ik + W_kj)
7:              end for
8:          end for
9:      end for
10:     for all k ∈ X do
11:         for all i ∈ S do
12:             for all j ∈ S do
13:                 W_ij := min(W_ij, W_ik + W_kj)
14:             end for
15:         end for
16:     end for
17:     FLOYD-WARSHALL(S, W)
18: end procedure
```

vertices, as shown in Fig. 4.2.2. It should be noted that we will make use of a sparsely connected graph, which simplifies the analysis. The result does not change for complete graphs, since the algorithm itself makes no such assumptions.

The first call to Floyd-Warshall (line 2) in Algorithm 13 finds the all-pairs shortest paths between the vertices in X, but using only vertices from X on the paths themselves. Note that there are two such sets shown in Fig. 4.2.2, i.e. X' and X'', with no direct edges between them. Thus, we can only find the shortest paths inside the individual sets. Once the paths are found, we can find our way from any vertex in X to any vertex in X if a path that does not take us through vertices in S exists.

The first loop block (lines 3 through 9) of Algorithm 13 finds every shortest path starting in X and ending in S, by going through vertices in X only. Every vertex in X knows the path to every other vertex in X, as long as the path does not go through vertices in S. At this point there must exist a pair of vertices $u \in X$, $v \in S$ where $W_{u,v} < \infty$ [1]. Thus, when the first loop block finishes, every vertex in X knows the shortest paths through X to some vertices in S. In Fig. 4.2.2 this means that the vertices in X' know the shortest paths through X' that end in S' or S''. The same is true for vertices in X''.

Finally, the second loop block (lines 10 through 16) of the algorithm finds every shortest path starting in some vertex in S, going through some vertex in X and ending in some vertex in S. The only vertices in S that have paths to vertices in X are those that have edges that connect them. However, the vertices in X that they are connected to, know the shortest paths through X ending in some vertices in S. Thus, the algorithm connects the sets S' and S'' via the shortest paths through X' and X''.

[1] If there were no such pair, a path from S to S going through X would not exist.

At the end (line 17), we run the Floyd-Warshall algorithm on S. Since the sets S' and S'' have been connected via shortest paths through X, we obtain the APSP solution for $S \times S$ whereby the paths can go through X.

Theorem 4.2.1 *Algorithm 13 computes the shortest paths between all pairs $S \times S$ in G.*

Proof See discusion above.

4.3 Experiments

For testing our strategies, described in Section 4.2.1 we used the real instances of the Traveling Visitor Problem, which were made from official tourist maps of cities of Koper, Belgrade and Venice. In the Belgrade example two different cases were considered and they differed in the size of the problem, i.e. the number of vertices in the graph. From the publicly available library, TSPLIB, of sample benchmarks for the TSP and related problems, two instances of the symmetric traveling salesman problem were selected, modified and tested.

These two instances were modified in such a way that a new graph G' was made satisfying the conditions of connected, weighted graph. Furthermore we split V into a set of vertices S and set of vertices X, such that $|S| = |X| = |V|/2$. A vertex degree 5 is arbitrarily assigned, inspired by the case of real instances, and means that from every vertex from V there is exactly 5 edges going to the other vertex from V. The 5 edges per vertex were chosen randomly, according to a uniform probability distribution.

Altogether 5 instances were tried out, with different sizes which range from 120 to 1002 vertices per instance. We compared two methods for solving the Traveling Visitor Problem. The first method is the Naïve algorithm, shown in Algorithm 10. The second tested method is the Koper algorithm, shown in Algorithm 11. For solving the TSP, as one step in both algorithms, we used the Concorde Algorithm, presented in Section 2.2.5. Furthermore, for solving the APSP, as a part of both algorithms, we used the Adapted Floyd−Warshall algorithm, which was presented in Section 4.2.2.

4.4 Results

The results of the experiment are summarized in Table 4.1. Five instances were tried out, with different sizes, ranging from 120 to 1002 vertices per instance. The names of these instances are in the first column. The second column contains the size of the problem, i.e. the number of vertices. The third column in Table 4.1 corresponds to the number of vertices in set S and in the top three instances the number of interesting sites from tourist maps. The fourth column contains names of the two tested methods. The fifth column corresponds to the length of the tour i.e., the cost of a solution, which was obtained in the experiment. In all six cases, Koper Algorithm obtains the shortest tours. The last column corresponds to the quality of

Name	(V)	(S)	Methods	Tour Cost	Difference
Koper	120	55	Naïve	4738	17.22%
			Koper	4042	
Belgrade	163	53	Naïve	100389	6.52%
			Koper	94246	
	250	90	Naïve	122119	8.77%
			Koper	112275	
Venice	210	72	Naïve	26648	24.24%
			Koper	21448	
lin318	318	159	Naïve	921499	249.08%
			Koper	263983	
pr1002	1002	501	Naïve	11818732	354.46%
			Koper	2600585	

Table 4.1: Two techniques for solving the Traveling Visitor Problem

the result. The comparison of both algorithms is computed by the relative difference of cost solutions that is defined as

$$q_A = \frac{l_A - l_C}{l_C} \times 100\% \tag{4.2}$$

where l_C is a length of the tour obtained by Koper algorithm and l_A is a length of the tour obtained by Naïve algorithm. The relative difference is expressed in percents, where, for example, 17.22% difference means that Naïve algorithm performed 17.22% worse than Koper algorithm. The first tested method, the Naïve algorithm, performed poorly in comparison to the Koper algorithm. The quality differs from 6.52% in the case of Belgrade163 to 354.46% in the case of pr1002 instance. The difference in the quality of the solutions of two tested algorithms grows with instances that are more complex.

Although these algorithms are similar in terms of components (both rely on solving an APSP and TSP), the difference on the quality of the solutions indicates that there is a gain by using the Koper algorithm. Note also that the running time for both algorithms is the same.

4.5 Conclusions

The goal of this chapter is to describe a new problem from graph theory, named the Traveling Visitor Problem. Although the new problem is similar to the Traveling Salesman Problem, when we try to solve it with the Naïve algorithm we get solutions far from optimal. The minimum cost solutions for the Traveling Visitor Problem instances tested in the chapter are provided by our Koper algorithm. The tested benchmarks are obtained from three real instances coming from tourist maps of cities of Koper, Belgrade and Venice and two modified instances from TSPLIB. In

all tested cases the Koper algorithm significantly outperforms the Naïve algorithm for solving the Traveling Visitor Problem.

The chapter opens a number of interesting questions for future research. The first one is related to the size of the problem. Namely, how will some heuristic algorithm behave in cases, where Concorde can no longer compute the optimal solution? A very interesting question is also how to apply our technique to Asymmetric Traveling Visitor Problem.

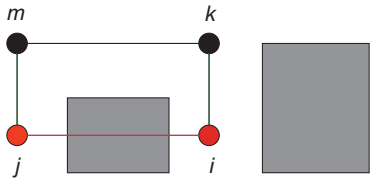

Figure 4.1: TSP and TVP, Two rectangles represent buildings (obstacles) in the city. Red nodes represent interesting sites in the city (vertices from set S), black nodes represent crossroads in the city (vertices from set X), the red line represent the Euclidean shortest connection between two interesting sites (this is the case in TSP), black lines represent the connection between two interesting sites, going through two crossroads (this is the case in TVP)

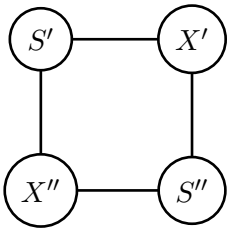

Figure 4.2: Each node in the graph represents an arbitrary amount of vertices from a single set that are arbitrarily interconnected. The edges represent (arbitrarily many) connections to other such sets. Note, that $S', S'' \subset S$ and $X', X'' \subset X$.

Chapter 5

Conclusion

The goal of the Thesis was to investigate influence of grafting a 2-opt based local searcher into the standard genetic algorithm, for solving the Traveling Salesman Problem. It is known that genetic algorithms are very successful when implemented for many NP-hard problems. However, they are much more effective if some specific knowledge about particular problem is utilized. The TSP is well researched problem with many such improvements, especially when the restricted version of the problem with Euclidean distance is considered. In our experiment in Chapter 3 we compared two direct techniques, with our grafted genetic algorithms. Solutions from Concorde and greedy algorithm were added for better comparison. Quantitative results on test cases from *TSPLIB* show that grafted algorithms have advantages. Even when both components have serious drawbacks, their grafted combinations exhibit very good behaviour. Results on examples from *TSPLIB* show that this method combines good qualities from both methods applied and significantly outperforms each individual method. In the second part of the experiment in Chapter 3 an influence of partial grafting a 2-opt local searcher into genetic algorithm was studied. The best performance was achieved in a configuration with 90% frequency use of local searcher. In a comparison with a performance of GGAemc the same quality of results was achieved in a shorter time, on average a 7% of running time was spared. The cases with 10% and 20% frequency use of local search provides fast and far from optimal solutions but still better then the GAemc and GAdpc. The configurations with 50% frequency use of local searcher present a good examples of trade-off between a running time and quality, especcialy in setting with ending sequence of local searcher. The best gain is attained when a local searcher is used in an ending sequence of the algorithm and in frequency not less then 50% and not more than 90%.

The second goal of the Thesis was to describe a new problem from graph theory, named the Traveling Visitor Problem. Although the new problem is similar to the Traveling Salesman Problem, when we try to solve it with the Naïve algorithm we get solutions far from optimal. The minimum cost solutions for the Traveling Visitor Problem instances tested in the paper are provided by Koper Algorithm. The tested benchmarks are combined from three real instances made using tourist maps of cities of Koper, Belgrade and Venice and two instances of modified cases from TSPLIB. In all tested cases the Koper Algorithm significantly outperforms the Naïve Algorithm

for solving the Traveling Visitor Problem.

The results of doctoral dissertation represents the contribution to bridging the gap between theoretical computer science and its application in practice. Also to better understanding and modeling of real problems in the economy, represented as the NP-hard problems from graph theory as well as a contribution to the optimization methods for solving these hard problems.

Povzetek v slovenskem jeziku

5.1 Uvod

Ko sem prišel leta 2007 na podiplomski doktorski študij računalništva, iz Beograda v Koper, sem prišel z veliko željo, da spoznam mesto, v katerem bom preživel naslednja štiri leta. Mestno jedro mi je bilo takoj všeč predvsem zaradi številnih znamenitosti, natančno 55, ki se nahajajo tudi na turističnem zemljevidu Kopra. Ker je doktorski študij naporen in nisem imel veliko prostega časa, sem se začel spraševati kako bi bilo, če bi lahko svoj obhod po mestnih znamenitostih optimiziral na tak način, da porabim najmanj možnih korakov in tako prihranim nekaj časa. Problem sem poimenoval *problem potujočega obiskovalca* (angl. *Traveling Visitor Problem, (TVP)*).

V zgodnjih 30. letih 20. stoletja, je avstrijski matematik Karl Menger izzval takratno raziskovalno skupnost, naj z matematičnega vidika preuči, sledeč problem [100]: Glasnik želi obiskati vsako mesto s seznama na katerem je n mest natanko enkrat in se nato vrniti v svoje mesto pri tem so cene potovanj iz mesta i v mesto j znane vnaprej. Vprašanje je torej kateri obhod je najcenejši? Problem Trgovskega Potnika (angl. *Traveling Salesman Problem, (TSP)*) je formalno definiran na polnemu grafu $G = (V, E)$, kjer je $V = \{v_1, v_2, ..., v_n\}$ množica vozlišč, E množica povezav in s cenilno funkcijo $c(i, j)$, ki povezavi $(i, j) \in E$ priredi določeno ceno.

TSP lahko obravnavamo tudi kot problem permutacij. Naj bo P_n množica vseh permutacij iz množice $\{1, 2, ..., n\}$. Potem je problem trgovskega potnika poiskati $\pi = (\pi(1), \pi(2), ..., \pi(n))$ v P_n, za katero velja, da je $c_{\pi(n)\pi(1)} + \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)}$, minimalen.

TSP je eden izmed najpomembnejših predstavnikov večje množice problemov, imenovane kombinatorični optimizacijski problemi [67]. Ker spada TSP v razred NP-težkih (angl. *NP-Hard*) problemov [76], učinkovitega algoritma za TSP ne poznamo. Natančneje, takšen algoritem obstaja če in samo če sta razreda P in NP enaka. S praktičnega vidika to pomeni, da ne poznamo natančnega algoritma za katerikoli TSP primer z n vozlišči, ki se obnaša značilno bolje, kot algoritem, ki izračuna vseh $(n-1)!$ možnih obhodov ter vrne obhod z najmanjšo ceno.

V praksi lahko za reševanje tega problema uporabimo tudi drugačen pristop. Določeni TSP primer, z n vozlišči ima lahko katerikoli obhod, ki poteka skozi vsa vozlišča n in predstavlja možno rešitev, ki je zgornja meja (angl. *upper bound*) za

najnižjo možno ceno. Algoritem, ki v polinomskem času (angl. *polynominal time*) konstruira možne rešitve s to zgornjo mejo se imenuje hevristika [12,121]. Načeloma, ti algoritmi tvorijo rešitve, vendar brez zagotovila o kakovosti rešitve glede na razliko med njihovo ceno in optimalno ceno.

Poznamo dve vrsti TSP: simetrični TSP in asimetrični TSP. V simetrični obliki, znani pod imenom STSP [73, 74, 93, 97], je razdalja med vozliščema i in j enaka razdalji med vozliščema j in i. V primeru asimetričnega TSP (ATSP) [14,16,18,24], takšna simetrija ne obstaja. Poleg tega obstaja še vrsta različnih variacij TSP, ki so opisane in raziskane v literaturi ter predstavljene v doktorski disertaciji v Poglavju 2. Spodaj povzemam nekatere izmed njih.

Grupirani TSP (angl. *Clustered TSP*) [65], ozko-grlni TSP (angl. *Bottleneck TSP*) [63], posplošen TSP (angl. *Generalized TSP*) [62, 79], problem glasnika (angl. *Messenger Problem*) [100], ki je znan tudi kot problem izgubljenega prodajalca (angl. *Wondering Salesman Problem*) [78], problem zamenjave (angl. *The swapping problem*) [2], problem minimalne latentnosti (angl. *Minimum Latency Problem*) [11] znan tudi kot problem dostavljalca (angl. *Delivery Man Problem*) [63] ali problem potujočega mojstra (angl. *Traveling Repairman Problem*) [53], problem seizmičnih plovil (angl. *Seismic Vessel Problem*) [61], ki je posplošitev problema skladiščnega dvigala (angl. *Stacker Crane Problem*) [25], problem potujočega turnirja (angl. *Traveling Tournament Problem*) [42], problem lokacije objekta (angl. *Facility Location Problem*) [41]. In končno problem potujočega obiskovalca, ki je podrobno opisan, raziskovan in rešen v doktorski disertaciji v Poglavju 4, za ta problem ne poznamo nobenega vira v literaturi.

Prvi koraki v reševanju TSP so bili klasični poskusi. Te metode so sestavljene iz natančnih in hevrističnih algoritmov. Natančne metode, kot so presek ravnine (angl. *cutting planes*) [32], vejitev in povezovanje (angl. *branch and bound*) [27, 32], lahko optimalno rešijo relativno majhne probleme (v odvisnosti od velikosti n), med tem ko metode, kot so različne variante Lin-Kernighan algoritma [6, 46, 70, 79] in Concorde tehnike [3–5] nam dajo relativno dobre rezultate, tudi za večje probleme. Posamezni algoritmi, zasnovani na požrešnih principih, kot sta najbližji sosed (angl. *nearest neigbour*) [63] in vpeto drevo (angl. *spanning tree*) [69], se prav tako uporabljajo za reševanje TSP.

Natančne metode za reševanje TSP rezultirajo z eksponentnimi računskimi kompleksnostmi, tako da so v izogib obstoječim slabostim potrebne nove metode. Te metode vključujejo različne principe optimizacijskih tehnik, naravno orientirani optimizacijski algoritmi, populacijsko orientirani optimizacijski algoritmi, ter drugi. Različna bitja in naravni sistemi, ki se razvijajo v naravi so zanimivi in dragoceni izvori navdiha, za raziskovanje in ustvarjanje novih sistemov in algoritmov za reševanje TSP, ter njegovih variacij. Nekatere od teh metod so predstavljene v doktorski disertaciji v Poglavju 2. Naj jih naštejemo nekaj.

Evolutivno računanje (angl. *Evolutionary Computation*) [101, 106, 127, 136], genetski algoritmi (angl. *Genetic Algorithms*) [43, 51, 52, 103, 109, 117, 126, 130, 134, 135, 139, 140], memetični algoritmi (angl. *Memetic Algorithms*) [62, 87–89, 104, 113], sistemi mravelj (angl. *Ant Systems*) [39], simulirano ohlajanje (angl. *Simulated Annealing*) [84], in naposled *cepljeni genetski algoritmi* (angl. *Grafted Genetic Algorithms*) [35], [38], [36], [37]. Slednji predstavljajo vrsto hibridnih genetskih algo-

ritmov in so raziskani, podrobno opisani ter demonstrirani v doktorski disertaciji v Poglavju 3.

5.2 Vsebina disertacije

Doktorska disertacija po Pravilniku o pripravi in zagovoru doktorske disertacije na Univerzi na Primorskem, vsebuje naslednja poglavja:

- Zahvala
- Povzetek
- Kazalo vsebine
- Poglavje 1 - Uvod
- Poglavje 2 - Ozadje
  - 2.1 Problem Trgovskega Potnika
  - 2.2 Optimizacijski Algoritmi
- Poglavje 3 - Cepljeni Genetski Algoritmi
- Poglavje 4 - Problem Potujočega Obiskovalca
- Zaključek
- Literatura
- Kazalo
- Izjava

V poglavju *Ozadje* smo predstavili osnovne pojme trgovskega potnika in optimizacijskih algoritmov, s pomočjo katerih so lahko nadaljnja poglavja disertacije postala razumljiva tudi širšemu krogu bralcev. Preostali poglavji so namenjeni predstavitvi doseženih ciljev disertacije. Poglavja so razdeljena na več podpoglavij.

V doktorski disertaciji sta obdelani dve temi iz teoretičnega računalništva. Optimizacijsko hevristična metoda imenovana cepljeni genetski algoritmi (GGA) in kombinatorično optimizacijski problem, imenovan problem potujočega obiskovalca (TVP). Cilja doktorske disertacije sta naslednja:

Cepljeni genetski algoritmi: Cilj je pokazati kakovost dobljenih rešitev in hitrost izvajanja cepljenega genetskega algoritma, kadar se uporablja za probleme Simetričnih TSP-jev, ki so na voljo na svetovnem spletu, v obliki splošno priznanih ocenjevalnih primerov (angl. *benchmarks*), kot tudi dejanskih primerov, nastalih za problem potujočega obiskovalca.

Problem potujočega obiskovalca: Cilj je opisati in definirati problem iz realnega življenja, ustvariti realne primere za mesta v okolici in rešiti primere problemov z uporabo nove metode ter znanih metod, ki bodo skupaj prikazane v doktorski disertaciji. Raziskovalni cilj disertacije je dokazovanje spodaj navedenih Hipotez 1 in 2.

Hipoteza 1: Metoda za reševanje TSP, sestavljena iz dveh neodvisnih metod, genetskega algoritma in 2-opt hevristike, združuje kakovosti obeh metod na takšen način, da ju znatno prekaša, glede na kakovost rešitve.

Hipoteza 2: Glede na kakovost rešitve posebna metoda za reševanje problema potujočega obiskovalca, prekaša splošne algoritme za reševanje problema trgovskega potnika, ko jih uporabimo za reševanje problema potujočega obiskovalca.

5.3 Raziskava

5.3.1 Cepljeni Genetski Algoritmi

Botanično cepljenje je postopek, ko je tkivo prve rastline pritrjeno na tkivo druge rastline. Cepljenje lahko zmanjša čas cvetenja in skrajša čas rejskega programa. Lokalni iskalec je razširitev konvencionalnega genetskega algoritma, saj ne obstaja potreba po uporabi komponent genetskega algoritma. To omogoča optimizacijo posameznih genomov, izven evolucijskega procesa. V našem algoritmu po izvedeni rekombinaciji (vrstica 7 v Algoritmu 9), se lokalni iskalec uporablja za optimizacijo vsakega genoma potomcev (vrstica 8 v Algoritmu 9). Zaradi uporabe omenjene zunanje optimizacijske metode, genetski algoritem ni več čist, zato govorimo o cepljenem genetskem algoritmu [35], [38]. Omenjena oblika optimizacije se izvaja lokalno ter spreminja genom s pomočjo hevrističnega spreminjanja rešitve. 2-opt lokalni iskalec 2.2.3 je lokalna optimizacijska metoda za TSP, ki je bila vcepljena v standardni genetski algoritem (vrstica 8 v Algoritmu 9). Ta lokalna optimizacijska metoda, izvaja 2-opt hevristiko, ki izmenjuje povezave grafa z namenom zmanjšanja dolžine obhoda. Postopek izmenjave je sestavljen iz odstranjevanja dveh povezav iz trenutnega obhoda in ponovnega povezovanja na najboljši možen način, poglej Figuro 2.1.

V opravljenem poskusu smo testirali vpliv cepljenja algoritma lokalnega iskalca z genetskim algoritmom za reševanje problema trgovskega potnika. Za testiranje naše strategije in primerjave le te z ostalimi rešitvami, smo uporabili primere simetričnega problema trgovskega potnika, ki so na voljo na spletu, v knjižnici TSPLIB [122]. Uporablili smo 20 primerov, z različno kompleksnostjo in obsegom od 14 do 150 mest, Tabela 3.1. Primerjali smo našo metodo (Cepljeni Genetski Algoritem), katere predstavnika sta algoritma GGAemc in GGAdpc, s štirimi drugimi metodami. Za zgornjo mejo kakovosti rešitve smo uporabili požrešno hevristiko (angl. *Greedy Heuristic* 2.2.2), za spodnjo mejo smo uporabili globalni minimum, pridobljen s Concorde 2.2.5. Nato smo primerjali našo cepljeno metodo z 2-opt in genetskim algoritmom.

Rezultati eksperimenta so predstavljeni v Tabeli 3.1. Šesti stolpec v tabeli predstavlja rešitve našega cepljenega algoritma, ki je bil programiran s pomočjo križanja povezav (angl. *edge map crossover* 2.2.4), kot operator za rekombinacijo (GGAemc). V sedemnajstih od dvajset obravnavanih primerov je bila najdena optimalna rešitev,

preostali trije primeri odstopajo od optimalne rešitve za 0,01, 0,10 in 0,22 odstotka. Rešitve so bile najdene hitro, porabili smo med 0,6 in 15,2 sekund ter v relativno majhnem številu generacij. Sedmi stolpec v Tabeli 3.1, pripada našemu cepljenemu genetskemu algoritmu, ki vsebuje križanje ohranjanja razdalje (angl. *distance preserving crossover* 2.2.4), kot operator za rekombinacijo (GGAdpc). V enajstih od dvajset obravnavanih primerov je bila najdena optimalna rešitev, v preostalih devetih primerih so odstopanja od optimalne rešitve od 0,13 do 0,32 odstotka. V primerjavi z GGAemc, sta čas izvajanja in število generacij GGAdpc nekoliko manjša, posebej v nižjem predelu tabele, ki predstavlja kompleksnejše primere.

Kvantitativni rezultati na testnih primerih iz TSPLIB kažejo, da imata cepljena algoritma, GGAemc in GGAdpc prednosti. Kljub številnim pomankljivostim njunih komponent, se njune kombinacije cepljenja zelo dobro obnesejo. Rezultati primerov iz TSPLIB kažejo, da omenjena metoda cepljenja združuje dobre lastnosti iz obeh uporabljenih metod in občutno prekaša vsako izmed njiju.

5.3.2 Problem Potujocega Obiskovalca

Obiskovalci so prispeli v hotel v nekem novem mestu z željo, da obiščejo vse zanimivosti mesta natanko enkrat in se po ogledu vrnejo v hotel. Na ogled mesta se odpravijo peš - po ulicah, sprehajalnih zonah in poteh za pešce. Cilj je skrajšati pot obiskovalca.

Problem potujočega obiskovalca je izpeljan iz *problema trgovskega potnika* , pri čemer velja pravilo, da obiskovalec izbira samo med potmi, ki jih je možno prehoditi. To pomeni, da so evklidske razdalje [57, 113], kot jih poznamo v evklidskem TSP, v našem primeru napačne. Obiskovalci uporabljajo sprehajalne poti in območja za pešce, ki so različno dolge. Te omejitve določajo težo povezav, ki povezujejo vozlišča v grafu.

Definicija TVP je sledeča: Imamo graf $G = (V, E, c)$, kjer je množica vozlišč $V = S \cup X$ in $S \cap X = \emptyset$, kjer sta S zanimivosti mesta in X križišča, E množica povezav ter c cena potovanja. Cilj je poiskati najkrajši zaprt sprehod (angl. *closed walk*, poglavje 2.1.1) skozi vsa vozlišča S (glede na c) v grafu G, pri čemer se lahko sprehodimo skozi X.

Prva predlagana metoda za reševanje problema potujočega obiskovalca je Naivni algoritem (angl. *Naïve Algorithm*), prikazan v Algoritmu 10. V prvi vrstici psevdokode, lahko razberemo naslednje parametre: S pripada zanimivim lokacijam v mestu, X pripada križiščem, E pripada množici povezav, W pa predstavlja matriko povezav grafa G, $(S \cup X) \times (S \cup X)$. V prvem koraku algoritma je problem potujočega obiskovalca rešen kot primer problema trgovskega potnika. V naslednjem koraku, iz matrike povezav W izdelamo matriko povezav Z dimenzij $(S \times S)$, ki predstavlja rešitev problema najkrajših poti vseh parov (angl. *All-Pairs Shortest Paths Problem (APSP)*, poglavje 2.1.5). V zanki (od vrstice 6 do 8) je prikazana rešitev za TVP. Druga predlagana metoda za reševanje problema potujočega obiskovalca je algoritem Koper (angl. *Koper Algorithm*), prikazan v Algoritmu 11. Prva vrstica psevdokode vsebuje enake parametre kot Naivni algoritem. V prvem koraku poiščemo najkrajše poti med vsemi pari vozlišč množice S v grafu G. Vhodno matriko razdalj označimo z

W in izhodno matriko razdalj z Z. V naslednjem koraku, rešimo problem trgovskega potnika za matriko razdalj Z. Poleg tega smo dobili T, ki je rešitev za problem potujočega obiskovalca.

Za testiranje naše strategije smo uporabili primere problema potujočega obiskovalca, ki smo jih izdelali iz uradnih turističnih zemljevidov za mesta Koper, Beograd in Benetke. V primeru Beograda smo izdelali dva primera, ki se razlikujeta po velikosti problema, oziroma po številu vozlišč v grafu. Iz javno dostopne knjižnice, TSPLIB, smo si izbrali, spremenili in testirali dva primera problema simetričnega trgovskega potnika. Izvedli smo poskuse za 5 primerov, z različnimi velikostmi, ki so od 120 do 1002 vozlišč za posamezen primer.

Za reševanje problema potujočega obiskovalca smo primerjali dve metodi. Prva metoda je gore omenjeni Naivni algoritem, prikazan v Algoritmu 10. Druga metoda je algoritem Koper, prikazan v Algoritmu 11. Za reševanje TSP, ki je eden izmed korakov pri obema algoritmoma, smo uporabili algoritem Concorde, ki je predstavljen v poglavju 2.2.5. Za reševanje problema najkrajše poti vseh parov smo uporabili prilagojeni Floyd-Warshallov algoritem, ki je bil predstavljen v poglavju 4.2.2.

Rezultati poskusa so predstavljeni v Tabeli 4.1. Peti stolpec Tabele 4.1, predstavlja dolžino sprehoda (cena rešitve, ki smo jo dobili pri poskusu). Stolpec se imenuje cena sprehoda (angl. *tour cost*) in v vseh šestih primerih so najkrajši sprehodi pridobljeni z algoritmom Koper. Zadnji stolpec v Tabeli 4.1, predstavlja razliko med uporabljenimi metodami, prikazano v odstotkih. Prva metoda, naivni algoritem, se je odrezal veliko slabše od algoritma Koper. Kakovost rešitev varira v intervalu od 6,52 odstotka, v primeru Belgrade163, do 354,46 odstotka, v primeru pr1002.

Namen tega raziskovanja je bil opis in rešitev novega problema v teoriji grafov, imenovanega problem potujočega obiskovalca. Čeprav je nov problem podoben dobro znanemu problemu trgovskega potnika, ko ga poskušamo rešiti z Naivnim algoritmom, so rezultati daleč od optimalnih. V vseh testiranih primerih problema potujočega obiskovalca, algoritem Koper občutno prekaša Naivni algoritem.

5.4 Metodologija

Glavno orodje za opis in definicijo problema potujočega obiskovalca je teorija grafov. Za realistično predstavitev vozlišč našega grafa, kot tudi povezav ter cen je uporabljen geografski informacijski sistem "Google Earth", čigar podatkovo bazo smo uporabili za pridobitev mestnih znamenitosti, križišč in sprehajalnih poti. Pomembno vlogo za dokazovanje hipoteze 1 ima platforma za raziskovanje genetskih algoritmov "EA Visualizer" [15], aplikacija napisana v programskem jeziku Java.

Za dokazovanje hipoteze 2 smo uporabili in ustrezno nadgradili Floyd-Warshallov algoritem [28], za iskanje najkrajših poti med vsemi pari vozlišč grafa $G = (V, E, c)$. Uporabili smo tudi znane optimizacijske metode za reševanje simetričnega in asimetričnega problema trgovskega potnika, presek-ravnine [114, 116], na čigar osnovi temeljijo metode Concorde [3–5] in hevristične metode Lin-Kernighan algoritma [6,46,70,79]. Obe aplikaciji sta pisani v programskem jeziku AnsiC. Poleg tega za dokazovanje hipoteze 2 smo uporabili znanje iz matematičnih modelov znanih kot problem linearnega programiranja (angl. *linear programming problems*) [23,132], kot

tudi simpleks metode (angl. *simplex methods*) [32].

5.5 Doprinos k znanosti

Doprinos k znanosti predstavljajo naslednji rezultati:

- Izdelava cepljenega genetskega algoritma za reševanje problema trgovskega potnika.
- Potrditev, da problem trgovskega potnika lahko uspešno rešimo z uporabo cepljenega genetskega algoritma.
- Izdelava posebne metode za reševanje problema potujočega obiskovalca.
- Izdelava realnih primerov problema potujočega obiskovalca za mesta Koper, Beograd in Benetke.
- Potrditev da vsak primer problema potujočega obiskovalca, ki je rešen z posebno metodo, predstavlja zelo zadovoljivo rešitev.

Rezultati doktorske disertacije predstavljajo doprinos k premoščanju razlike med teoretičnim računalništvom in njegovo uporabo v praksi: boljšemu razumevanju in modeliranju realnih problemov iz gospodarstva, predstavljenih kot NP-težki problemi v teoriji grafov, kot tudi doprinos optimizacijskim metodam za reševanje omenjenih problemov.

Naj omenimo še, da so rezultati disertacije objavljeni v naslednjih znanstvenih člankih:

- M. Djordjevic, Influence of Grafting a Hybrid Searcher Into the Evolutionary Algorithm, *Proceedings of the 17th International Electrotechnical and Computer Science Conference, Portoroz, Slovenia* (2008), 115–118.
- M. Djordjevic, and M. Tuba, and B. Djordjevic, Impact of Grafting a 2-opt Algorithm Based Local Searcher Into the Genetic Algorithm, *Proceedings of the 9th WSEAS international conference on Applied informatics and communications, AIC 2009, Moscow, Russia* (2009), 485–490.
- M. Djordjevic, and A. Brodnik, Quantitative Analysis of Separate and Combined Performance of Local Searcher and Genetic Algorithm, *Book of Abstract of International Conference on Operations Research, OR 2011, Zurich, Switzerland* (2011), 130.
- M. Djordjevic, and A. Brodnik, Quantitative Analysis of Separate and Combined Performance of Local Searcher and Genetic Algorithm, *Proceedings of the 33rd International Conference on Information Technology Interfaces, ITI 2011, Dubrovnik, Croatia* (2011), 515–520.
- M. Djordjevic, A. Brodnik and M. Grgurovic, The Traveling Visitor Problem and Koper Algorithm for Solving It, accepted by *25th Conference of European Chapter on Combinatorial Optimization, ECCO2012, Antalya, Turkey.*

- M. Djordjevic, A. Brodnik and M. Grgurovic, The Traveling Visitor Problem and Algorithms for Solving It, accepted by *3rd Student Conference on Operational Research, SCOR 2012, Nottingham, UK.*

Bibliography

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice hall, Upper Saddle River (New Jersey), 1993.

[2] S. Anily and R. Hassin. The swapping problem. *Networks*, 22(4):419–433, 1992.

[3] D. Applegate, R. Bixby, V. Chvátal, and B. Cook. Finding cuts in the TSP. Technical report, Center for Discrete Mathematics and Theoretical Computer Science, Rutgers, 1995.

[4] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. TSP cuts which do not conform to the template paradigm. In M. Jünger and D. Naddef, editors, *Computational Combinatorial Optimization*, pages 261–304. Springer Verlag, Berlin, 2001.

[5] D. Applegate, R. Bixby, W. Cook, and V. Chvátal. *On the solution of traveling salesman problems*. Rheinische Friedrich-Wilhelms-Universitat Bonn, Bonn, 1998.

[6] D. Applegate, W. Cook, and A. Rohe. Chained Lin-Kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 15(1):82–92, 2003.

[7] T. Bäck. *Evolutionary algorithms in theory and practice*. Oxford University Press, New York, 1996.

[8] M. O. Ball and M. J. Magazine. Sequencing of insertions in printed circuit board assembly. *Operations Research*, 36(2):192–201, 1988.

[9] J. C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing*, 6(2):154–160, 1994.

[10] J. J. Bentley. Fast algorithms for geometric traveling salesman problems. *INFORMS Journal on Computing*, 4(4):387, 1992.

[11] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan. The minimum latency problem. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 163–171, 1994.

[12] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003.

[13] K. D. Boese. *Models for iterative global optimization.* PhD thesis, University of California, Los Angeles, 1996.

[14] N. Boland, L. Clarke, and G. Nemhauser. The asymmetric traveling salesman problem with replenishment arcs. *European Journal of Operational Research*, 123(2):408–427, 2000.

[15] P. Bosman and D. Thierens. On the modelling of evolutionary algorithms. In *Proceedings of the Eleventh Belgium–Netherlands Conference on Artificial Intelligence BNAIC*, pages 67–74, 1999.

[16] J. Brest and J. Žerovnik. A heuristic for the asymmetric traveling salesman problem. In *The 6th Metaheuristics International Conference*, pages 145–150, 2005.

[17] T. N. Bui and B. R. Moon. A new genetic approach for the traveling salesman problem. In *Evolutionary Computation, IEEE World Congress on Computational Intelligence, Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 7–12, 1994.

[18] G. Carpaneto, M. Dell'Amico, and P. Toth. Exact solution of large-scale, asymmetric traveling salesman problems. *ACM Transactions on Mathematical Software (TOMS)*, 21(4):394–409, 1995.

[19] A. E. Carter and C. T. Ragsdale. A new approach to solving the multiple traveling salesperson problem using genetic algorithms. *European journal of operational research*, 175(1):246–257, 2006.

[20] R. Cheng and M. Gen. Resource constrained project scheduling problem using genetic algorithms. *International Journal of Intelligent Automation and Soft Computing*, 3(3):78–286, 1997.

[21] R. Cheng, M. Gen, and Y. Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms - i. representation. *Computers & Industrial Engineering*, 30(4):983–997, 1996.

[22] R. Chiong. *Nature-inspired algorithms for optimisation.* Springer Verlag, Berlin, 2009.

[23] V. Chvátal. *Linear programming.* WH Freeman, New York, 1983.

[24] J. Cirasella, D. Johnson, L. McGeoch, and W. Zhang. The asymmetric traveling salesman problem: Algorithms, instance generators, and tests. In *Revised Papers from the Third International Workshop on Algorithm Engineering and Experimentation*, ALENEX '01, pages 32–59, 2001.

[25] A. Coja-Oghlan, S. O. Krumke, and T. Nierhoff. A heuristic for the stacker crane problem on trees which is almost surely exact. *Journal of Algorithms*, 61(1):1–19, 2006.

[26] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, 1971.

[27] W. Cook and P. Seymour. Tour merging via branch-decomposition. *INFORMS Journal on Computing*, 15(3):233–248, 2003.

[28] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. 3rd edition. The MIT Press, Cambridge, 2009.

[29] G. Cornuéjols, J. Fonlupt, and D. Naddef. The traveling salesman problem on a graph and some related integer polyhedra. *Mathematical Programming*, 33(1):1–27, 1985.

[30] G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, 1958.

[31] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954.

[32] G. B. Dantzig and W. Orchard-Hays. The product form for the inverse in the simplex method. *Mathematical Tables and Other Aids to Computation*, 8(46):64–67, 1954.

[33] S. Dasgupta, C. H. Papadimitriou, and U. Vazirani. *Algorithms*. 1st ed. McGraw-Hill, New York, 2008.

[34] F. Della Croce, R. Tadei, and G. Volta. A genetic algorithm for the job shop problem. *Computers & Operations Research*, 22(1):15–24, 1995.

[35] M. Djordjevic. Influence of grafting a hybrid searcher into the evolutionary algorithm. In *Proceedings of the 17th International Electrotechnical and Computer Science Conference*, pages 115–118. Slovenian Section IEEE, Ljubljana, 2008.

[36] M. Djordjevic and A. Brodnik. Quantitative analysis of separate and combined performance of local searcher and genetic algorithm. In *Book of Abstracts, OR 2011, International Conference on Operation Research*, page 130. IFOR, Zurich, 2011.

[37] M. Djordjevic and A. Brodnik. Quantitative analysis of separate and combined performance of local searcher and genetic algorithm. In *Proceedings of the 33rd International Conference on Information Technology Interfaces, ITI 2011*, pages 515–520. SRCE, Zagreb, 2011.

[38] M. Djordjevic, M. Tuba, and B. Djordjevic. Impact of grafting a 2-opt algorithm based local searcher into the genetic algorithm. In *Proceedings of the 9th WSEAS international conference on Applied informatics and communications*, pages 485–490. Stevens Point, WSEAS, 2009.

[39] M. Dorigo, M. Birattari, and T. Stutzle. Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39, 2006.

[40] M. Dorigo and L. M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on*, 1(1):53–66, 1997.

[41] Z. Drezner. *Facility location: a survey of applications and methods*. Springer, Berlin, 1995.

[42] K. Easton, G. Nemhauser, and M. Trick. The traveling tournament problem description and benchmarks. In *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, CP '01, pages 580–584, 2001.

[43] T. A. El-Mihoub, A. A. Hopgood, L. Nolle, and A. Battersby. Hybrid genetic algorithms: a review. *Engineering Letters*, 13(2):124–137, 2006.

[44] H. Emmons and K. Mathur. Lot sizing in a no-wait flow shop. *Operations research letters*, 17(4):159–164, 1995.

[45] C. Engels and B. Manthey. Average-case approximation ratio of the 2-opt algorithm for the tsp. *Operations Research Letters*, 37(2):83–84, 2009.

[46] T. Fischer and P. Merz. A distributed chained Lin-Kernighan algorithm for TSP problems. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, pages 162–172, 2005.

[47] B. Fleischmann. A cutting plane procedure for the travelling salesman problem on road networks. *European Journal of Operational Research*, 21(3):307–317, 1985.

[48] C. Fleurent and J. A. Ferland. Genetic hybrids for the quadratic assignment problem. *American Mathematical Society*, 16:173–187, 1993.

[49] M. M. Flood. The traveling-salesman problem. *Operations Research*, 4(1):61–75, 1956.

[50] L. J. Fogel. *Intelligence through simulated evolution: forty years of evolutionary programming*. John Wiley & Sons, New York, 1999.

[51] B. Freisleben and P. Merz. New genetic local search operators for the traveling salesman problem. In *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, PPSN IV, pages 890–899, 1996.

[52] P. Gang, I. Iimura, and S. Nakayama. An evolutionary multiple heuristic with genetic local search for solving TSP. *International Journal of Information Technology*, 14(2):1–11, 2008.

[53] A. Garcia, P. Jodrá, and J. Tejel. A note on the traveling repairman problem. *Networks*, 40(1):27–31, 2002.

[54] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman, New York, 1979.

[55] I. Gerace and F. Greco. The travelling salesman problem in symmetric circulant matrices with two stripes. *Mathematical Structures in Computer Science*, 18(01):165–175, 2008.

[56] F. Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65(1-3):223–253, 1996.

[57] M. X. Goemans and D. J. Bertsimas. Probabilistic analysis of the Held and Karp lower bound for the Euclidean traveling salesman problem. *Mathematics of operations research*, 16(1):72–89, 1991.

[58] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, Reading, 1989.

[59] L. Gouveia and S. Vob. A classification of formulations for the (time-dependent) traveling salesman problem. *European Journal of Operational Research*, 83(1):69–82, 1995.

[60] A. Gupta, V. Nagarajan, and R. Ravi. Approximation algorithms for optimal decision trees and adaptive tsp problems. In *Proceedings of the 37th International Colloquium Conference on Automata, languages and programming*, ICALP'10, pages 690–701, 2010.

[61] G. Gutin, H. Jakubowicz, S. Ronen, and A. Zverovitch. Seismic vessel problem. *Communications in Dependability and Quality Management*, 8(1):13–20, 2005.

[62] G. Gutin and D. Karapetyan. A memetic algorithm for the generalized traveling salesman problem. *Natural Computing*, 9(1):47–60, 2010.

[63] G. Gutin and A. P. Punnen. *The traveling salesman problem and its variations*. Kluwer Academic Publishers, Dordrecht, 2002.

[64] G. Gutin, A. Yeo, and A. Zverovitch. Exponential neighborhoods and domination analysis for the tsp. In D. Du, P. Pardalos, G. Gutin, and A. Punnen, editors, *The traveling salesman problem and its variations*, pages 223–256. Kluwer Academic Publishers, Dordrecht, 2004.

[65] N. Guttmann-Beck, R. Hassin, S. Khuller, and B. Raghavachari. Approximation algorithms with bounded performance guarantees for the clustered traveling salesman problem. *Algorithmica*, 28(4):422–437, 2000.

[66] M. Hahsler and K. Hornik. TSP–Infrastructure for the traveling salesperson problem. *Journal of Statistical Software*, 23(2):1–21, 2007.

[67] W. E. Hart. *Adaptive global optimization with local search*. PhD thesis, University of California, San Diego, 1994.

[68] L. He and N. Mort. Hybrid genetic algorithms for telecommunications network back-up routeing. *BT Technology Journal*, 18(4):42–50, 2000.

[69] M. Held and R.M. Karp. The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 1(1):6–25, 1971.

[70] K. Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[71] J. H. Holland. *Adaptation in natural and artificial systems*. The University of Michigan Press, Ann Arbor, 1975.

[72] H. Hoos and T. Stützle. *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann Publishers, San Francisco, 2004.

[73] L. J. Hubert and F. B. Baker. Applications of combinatorial programming to data analysis: the traveling salesman and related problems. *Psychometrika*, 43(1):81–91, 1978.

[74] D. Johnson and L. McGeoch. Experimental analysis of heuristics for the STSP. In *The traveling salesman problem and its variations*, pages 369–443. Kluwer Academic Publishers, Dordrecht, 2004.

[75] D. S. Johnson and L. A. McGeoch. The traveling salesman problem: a case study in local optimization. In *Local search in combinatorial optimization*, pages 215–310. J. Wiley, Chichester, 1997.

[76] D. S. Johnson and C. H. Papadimitriou. *Computational complexity and the traveling salesman problem*. Massachusetts Institute of Technology, Cambridge, 1981.

[77] S. Jung and B. R. Moon. The natural crossover for the 2d euclidean tsp. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*, pages 1003–1010. Morgan Kaufmann, San Mateo, 2000.

[78] M. Junger, G. Reinelt, and G. Rinaldi. The traveling salesman problem. *Handbooks in Operations Research and Management Science*, 7:225–330, 1995.

[79] D. Karapetyan and G. Gutin. Lin-Kernighan heuristic adaptations for the generalized traveling salesman problem. *European journal of operational research*, 208(3):221–232, 2011.

[80] R. M. Karp and C. H. Papadimitriou. On linear characterizations of combinatorial optimization problems. *SIAM Journal on Computing*, 11:610–620, 1982.

[81] K. Katayama and H. Narihisa. Iterated local search approach using genetic transformation to the traveling salesman problem. In *Proceedings of GECCO'99*, pages 321–328, 1999.

[82] K. Katayama, H. Sakamoto, and H. Narihisa. The efficiency of hybrid mutation genetic algorithm for the travelling salesman problem. *Mathematical and Computer Modelling*, 31(10-12):197–203, 2000.

[83] R. Keuthen. *Heuristic Approaches for Routing Optimisation*. PhD thesis, University of Nottingham, Nottingham, 2003.

[84] S. Kirkpatrick. Optimization by simulated annealing: quantitative studies. *Journal of Statistical Physics*, 34(5):975–986, 1984.

[85] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671, 1983.

[86] J. R. Koza. *Genetic programming: On the programming of computers by natural selection*. MIT Press, Cambridge, 1992.

[87] N. Krasnogor. *Studies on the theory and design space of memetic algorithms*. PhD thesis, University of the West of England, Bristol, 2002.

[88] N. Krasnogor, P. Moscato, and M. G. Norman. A new hybrid heuristic for large geometric traveling salesman problems based on the delaunay triangulation. In *Anais do XXVII Simposio Brasileiro de Pesquisa Operacional, Vitoria, Brazil, SOBRAPO*, pages 6–8, 1995.

[89] N. Krasnogor and J. Smith. A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *Evolutionary Computation*, 9(5):474–488, 2005.

[90] K. W. C. Ku and M. W. Mak. Empirical analysis of the factors that affect the baldwin effect. *Lecture notes in computer science*, pages 481–490, 1998.

[91] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[92] L. T. Leng. Guided genetic algorithm. *Relation*, 10(148):103–113, 2008.

[93] J. K. Lenstra and A. H. G. R. Kan. Some simple applications of the travelling salesman problem. *Operational Research Quarterly*, 26(4):717–733, 1975.

[94] C. F. Liaw. A hybrid genetic algorithm for the open shop scheduling problem. *European Journal of Operational Research*, 124(1):28–42, 2000.

[95] G. F. Lima, A. S. Martinez, and O. Kinouchi. Deterministic walks in random media. *Physical Review Letters*, 87(1):10603, 2001.

[96] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations research*, 21(2):498–516, 1973.

[97] S. B. Liu, K. M. Ng, and H. L. Ong. A new heuristic algorithm for the classical symmetric traveling salesman problem. *International Journal of Mathematical Science*, 37(4):234–238, 2007.

[98] V. Mak and N. Boland. Heuristic approaches to the asymmetric travelling salesman problem with replenishment arcs. *International Transactions in Operational Research*, 7(4-5):431–447, 2000.

[99] R. Matai and S. Singh. Traveling Salesman Problem: an overview of applications, formulations and solution approaches. *Omega*, 34(3):209–219, 2006.

[100] K. Menger. Das botenproblem. *Ergebnisse eines Mathematischen Kolloquiums*, 2:11–12, 1932.

[101] M. Mernik, V. Žumer, and M. Črepinšek. A metaevolutionary approach for the traveling salesman problem. In *Information Technology Interfaces, ITI 2000, Proceedings of the 22nd International Conference*, pages 357–362, 2000.

[102] P. Merz. *Memetic algorithms for combinatorial optimization problems: Fitness landscapes and effective search strategies*. PhD thesis, University of Siegen, Siegen, 2000.

[103] P. Merz and B. Freisleben. Genetic local search for the TSP: new results. In *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)*, pages 159–164, 1997.

[104] P. Merz and B. Freisleben. Memetic algorithms for the traveling salesman problem. *Complex Systems*, 13(4):297–346, 2001.

[105] G. M. Morris, D. S. Goodsell, R. S. Halliday, R. Huey, W. E. Hart, R. K. Belew, and A. J. Olson. Automated docking using a lamarckian genetic algorithm and an empirical binding free energy function. *Journal of computational chemistry*, 19(14):1639–1662, 1998.

[106] R. E. Mowe and B. A. Julstrom. A web-based evolutionary algorithm demonstration using the traveling salesman problem. In *The 34th Annual Midwest Instruction and Computing Symposium*, pages 1–10, 2001.

[107] Y. Nagata. Edge assembly crossoveria high-power genetic algorithm for the traveling salesman problem. In *Proceedings of the 7th International Conference on Genetic Algorithms*, page 450. Morgan Kaufmann Publishers, San Francisco, 1997.

[108] H. D. Nguyen, I. Yoshihara, K. Yamamori, and M. Yasunaga. Greedy genetic algorithms for symmetric and asymmetric tsps. *Joho Shori Gakkai Shinpojiumu Ronbunshu*, 2001(12):67–74, 2001.

[109] H. D. Nguyen, I. Yoshihara, K. Yamamori, and M. Yasunaga. Implementation of an effective hybrid GA for large-scale traveling salesman problems. *Systems, Manand Cybernetics*, 37(1):92–99, 2007.

[110] C. E. Noon and J. C. Bean. An efficient transformation of the generalized traveling salesman problem. In *Technical report*. University of Michigan, Ann Arbor, 1989.

[111] CS Orloff. A fundamental problem in vehicle routing. *Networks*, 4(1):35–64, 1974.

[112] A. J. Orman and H. P. Williams. A survey of different integer programming formulations of the travelling salesman problem. In *Optimisation, Econometric and Financial Analysis*, pages 91–104. Springer, Berlin, 2007.

[113] E. Ozcan and M. Erenturk. A brief review of memetic algorithms for solving Euclidean 2D traveling salesrep problem. In *Proceedings of the 13th Turkish Symposium on Artificial Intelligence and Neural Networks*, pages 99–108, 2004.

[114] M. Padberg and M. Grötschel. Polyhedral computations. In *The Traveling Salesman Problem*, pages 307–360. John Wiley & Sons, Chichester, 1985.

[115] M. Padberg and G. Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6(1):1–7, 1987.

[116] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

[117] S. S. Ray, S. Bandyopadhyay, and S. K. Pal. Genetic operators for combinatorial optimization in TSP and microarray gene ordering. *Applied Intelligence*, 26(3):183–195, 2007.

[118] I. Rechenberg. Evolution strategy. In *Computational Intelligence: imitating life*, pages 147–159. IEEE, New York, 1994.

[119] C. R. Reeves. A genetic algorithm for flowshop sequencing. *Computers & Operations Research*, 22(1):5–13, 1995.

[120] C. Rego. Relaxed tours and path ejections for the traveling salesman problem. *European Journal of Operational Research*, 106(2-3):522–538, 1998.

[121] C. Rego and F. Glover. Local search and metaheuristics. In *The traveling salesman problem and its variations*, pages 309–368. Kluwer Academic Publishers, Dordrecht, 2004.

[122] G. Reinelt. Tsplib-a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[123] G. Reinelt. *The traveling salesman: computational solutions for TSP applications*. Springer-Verlag, Berlin, 1994.

[124] A. Schrijver. On the history of combinatorial optimization (till 1960). *Handbooks in operations research and management science*, 12:1–68, 2005.

[125] V. Sels and M. Vanhoucke. A hybrid dual-population genetic algorithm for the single machine maximum lateness problem. In *Evolutionary Computation in Combinatorial Optimization*, volume 6622, pages 14–25. Springer Verlag, New York, 2011.

[126] H. Sengoku and I. Yoshihara. A fast TSP solver using GA on Java. In *3rd International Symposium on Artificial Life and Robotics (AROB III'98)*, 1998.

[127] W. Spears, K. De Jong, T. Bäck, and D. Fogel. An overview of evolutionary computation. In *Machine Learning: ECML-93*, pages 442–459. Springer, Heidelberg, 1993.

[128] K. Steiglitz and P. Weiner. Some improved algorithms for computer solution of the traveling salesman problem. In *Proceedings of 6th Annual Allerton Conference on Circuit and System Theory*, pages 814–821. University of Illinois, Urbana, 1968.

[129] T. Stützle and M. Dorigo. Aco algorithms for the traveling salesman problem. *Evolutionary Algorithms in Engineering and Computer Science*, pages 163–183, 1999.

[130] A. Uğur, S. Korukoğlu, A. Çalışkan, M. Cinsdikici, and A. Alp. Genetic algorithm based solution for TSP on a sphere. *Mathematical and Computational Applications*, 14(3):219–228, 2009.

[131] N. Ulder, E. Aarts, H. J. Bandelt, P. van Laarhoven, and E. Pesch. Genetic local search algorithms for the traveling salesman problem. In *Parallel problem solving from nature*, pages 109–116. Springer Verlag, Berlin, 1991.

[132] R. Vanderbei. Linear programming: foundations and extensions. *Journal of the Operational Research Society*, 49(1):93–98, 1998.

[133] M. Vazquez and L. D. Whitley. A hybrid genetic algorithm for the quadratic assignment problem. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*, pages 135–142. Morgan Kaufmann, San Mateo, 2000.

[134] M. B. Wall. *A Genetic Algorithm for Resource-Constrained Scheduling*. PhD thesis, Massachusetts Institute of Technology, Massachusetts, 1996.

[135] S. Wang, B. Wang, and X. Li. Grafted genetic algorithm and its application. In *7th International Conference on Computer-Aided Industrial Design and Conceptual Design, CAIDCD'06*, pages 1–4, 2007.

[136] C. M. White and G. G. Yen. A hybrid evolutionary algorithm for traveling salesman problem. *Evolutionary Computation*, 2(1):1473–1478, 2004.

[137] M. Yamamura, T. Ono, and S. Kobayashi. Character-preserving genetic algorithms for traveling salesman problem. *Journal-Japanese Society For Artificial Intelligence*, 7:1049–1049, 1992.

[138] M. Zachariasen and M. Dam. Tabu search on the geometric traveling salesman problem. In *Meta-Heuristics: Theory and Applications*, volume 36, pages 571–587. Kluwer Academic Publishers, Boston, 1996.

[139] F. Zhao, J. Sun, S. Li, and W. Liu. A hybrid genetic algorithm for the traveling salesman problem with pickup and delivery. *International Journal of Automation and Computing*, 6(1):97–102, 2009.

[140] G. Zhao, W. Luo, H. Nie, and C. Li. A genetic algorithm balancing exploration and exploitation for the travelling salesman problem. In *4th International Conference on Natural Computation*, pages 505–509, 2008.

Index

Declaration

I declare that this PhD Thesis does not contain any materials previously published or written by another person except where due reference is made in the text.

Milan Djordjevic