Algorithms for the Precedence Constrained Generalized Travelling Salesperson Problem

RAAD SALMAN

Department of Mathematical Sciences
Division of Mathematics
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2015
 Algorithms for the Precedence Constrained Generalized Travelling Salesperson Problem

Raad Salman
Algorithms for the Precedence Constrained Generalized Travelling Salesperson Problem

Raad Salman

August 27, 2015
Abstract

This thesis aims to implement and evaluate heuristic algorithms as well as formulating a MILP model for solving the precedence constrained travelling salesperson problem (PCGTSP). Numerical experiments are carried out on synthetic and real problem instances of different types. The MILP model is tested together with a generic solver to produce lower bounds on the optimal values of the problem instances. Results indicate that \( k \)-opt algorithms and metaheuristic algorithms that have been previously employed for related problems need further adaptation to be truly effective for the PCGTSP. The MILP model is shown to produce lower bounds of varying quality depending on the problem instance.
Acknowledgments

I would like to thank my supervisors Fredrik Ekstedt and Ann-Brith Strömberg for their guidance during this thesis work as well as Domenico Spensieri, Johan Torstensson and everyone else who helped me during my time at the Fraunhofer-Chalmers Research Centre.
## Contents

1 Introduction ................................. 1  
  1.1 Background .................................. 1  
  1.2 Motivation .................................. 2  
  1.3 Limitations .................................. 2  
  1.4 Outline .................................... 3  

2 Problem Description ......................... 4  
  2.1 Related Problems ......................... 4  
      2.1.1 The Sequential Ordering and the Precedence Constrained Asymmetrical  
            Travelling Salesperson Problems .................................. 5  
      2.1.2 The Generalized Travelling Salesperson Problem ....................... 5  
  2.2 The Precedence Constrained Generalized Travelling Salesperson Problem .... 6  

3 Literature Review ............................. 7  
  3.1 The Sequential Ordering Problem & The Precedence Constrained Asymmetric  
      Travelling Salesperson Problem .................................. 7  
  3.2 The Generalized Travelling Salesperson Problem ............................... 8  
  3.3 The Precedence Constrained Generalized Travelling Salesperson Problem .... 8  

4 Integer Linear Programming ................... 9  
  4.1 Linear Optimization .......................... 9  
      4.1.1 Computational complexity .......................... 10  
      4.1.2 Convexity .................................. 10  
      4.1.3 Linearity .................................. 11  
      4.1.4 Relations between Linear Programming and Integer Linear Programming 11  
  4.2 Optimizing Algorithms ...................... 12  
      4.2.1 Cutting Plane Methods .................................. 12  
      4.2.2 Branch-and-Bound .................................. 13  
      4.2.3 Branch-and-Cut .................................. 14  
      4.2.4 Dynamic Programming ................................. 14  
  4.3 Heuristic Algorithms ...................... 15  
      4.3.1 Construction Heuristics .................................. 15  
      4.3.2 Local Search Heuristics .................................. 15  
      4.3.3 Metaheuristics .................................. 16  

5 Mathematical Modelling ....................... 17  
  5.1 Notation .................................. 17  
  5.2 The Model GTSP1 ............................ 17  
  5.3 The Model PCATSP1 ......................... 18  
  5.4 A proposed PCGTSP model .................... 19
1

Introduction

1.1 Background

Within the field of automation, robot station optimization is an important subject where many different subproblems arise. Some of these are path optimization, station load balancing and robot coordination (see [1]). The problem of optimizing task sequences occurs naturally in conjunction with path optimization and is often modelled as a travelling salesperson problem or some variation of it.

An application of such a task sequence optimization problem is the case of using coordinate measuring machines (CMMs) to measure a series of points on an object (see Figure 1.1) with the purpose of determining the object’s geometrical characteristics (see [2]). A CMM consists of a robot arm capable of moving along three orthogonal axes and a motorized probe head which is capable of moving 360 degrees horizontally and 180 degrees vertically. Since a CMM has five degrees of freedom each point can be approached from many different angles and thus be measured in a multitude of ways. Furthermore, the order in which the points are measured may have an effect on the quality of the measurement results. To model these characteristics, one can discretize a subset of the different ways in which a point can be measured and constrain the order of the points being evaluated. Given such a discretization and set of constraints one can model the problem of minimizing the total measuring time as a precedence constrained generalized travelling salesperson problem (PCGTSP). Similar modelling can be done in other applications such as automated robot welding.
1.2 Motivation

The purpose of this thesis work is to develop, implement, and evaluate algorithms for solving the PCGTSP. The thesis also aims to formulate a strong mathematical optimization model capable of producing tight lower bounds and to study relevant literature which deals with the PCGTSP and related problems, such as the generalized travelling salesperson problem (GTSP), the sequential ordering problem (SOP) and the precedence constrained asymmetric TSP (PCATSP). Even though these related optimization problems have been extensively studied, very few studies have been made on the particular subject of the PCGTSP (see [3] and [4]). To the best of our knowledge, no attempt at a thorough investigation of the PCGTSP has been made. The main goal of this thesis is to make such an investigation and to develop fast and effective algorithms for solving the PCGTSP.

1.3 Limitations

This thesis will focus on evaluating different heuristic algorithms. The reason is that for large problems optimizing algorithms are likely to take an unreasonable amount of time to arrive at a solution. The mixed integer linear programming (MILP) model formulated in this thesis is used together with a generic solver to find optimal values or lower bounds for the different problem instances that are used for the computational experiments. By doing this, the quality of the bounds produced by the MILP model is evaluated and the results from the heuristic algorithms can be more objectively assessed in comparison to the bounds. The quality of the bounds produced by the MILP model are crucial for the viability of future development of problem specific optimizing algorithms. Development of such optimizing algorithms could in turn involve a more extensive investigation of the polytope defined by the convex hull of the set of feasible solutions to the MILP model and the strengthening of the model by the development of valid inequalities (see [5, Ch. II.2 and II.5]).
1.4 Outline

In Chapter 2 an introduction to the PCGTSP and its related problems is presented. In Chapter 3 relevant research which relates to the PCGTSP, the GTSP and the SOP/PCATSP is reviewed. In Chapter 4 some basic mathematical concepts and different classifications of problems and algorithms are described. In Chapter 5 the proposed MILP formulation of the PCGTSP is presented together with relevant mathematical notation. Chapter 6 outlines and describes the algorithms which are implemented and evaluated. Chapters 7 and 8 contain the methodology behind the experiments and the results. In Chapter 9 the results are discussed and in Chapter 10 suggestions for future research are given.
2

Problem Description

In this chapter we describe the PCGTSP, its connection to the related problems TSP, GTSP and SOP/PCATSP and present the particular challenges of the PCGTSP.

2.1 Related Problems

The travelling salesperson problem (TSP) is an optimization problem which poses the question: "Given a set of cities and all the distances between them, what is the shortest route if each city is to be visited exactly once?". In a more abstract sense one can imagine the TSP as defined on a set of \textit{nodes} and a set of \textit{edges} which connect them. Each edge is associated with some cost. The problem is then to construct a solution in which each node is visited exactly once and one returns to the start node such that the sum of the arc-costs is minimized (see [6, p. xi]). A feasible (but not necessarily optimal) solution to the TSP is referred to as a \textit{tour}.

![Figure 2.1: An example of a solution to a TSP instance. Bold edges are traversed.](image)

This famous optimization problem has many variations and a wide range of applications (see [6, p. xi]). To accurately describe the PCGTSP we need to present some variations of the TSP which have been studied previously and which feature similar challenges as the PCGTSP.
2.1. RELATED PROBLEMS

2.1.1 The Sequential Ordering and the Precedence Constrained Asymmetric Travelling Salesperson Problems

The SOP, like the TSP, is defined on a set of nodes and edges. However, the SOP defines a fixed start node and a fixed end node. A feasible solution for the SOP is then a path between the start node and the end node such that each node is visited exactly once (see e.g. [7,8]). The PCATSP is also defined on a set of nodes and edges but like the TSP it requires a closed tour where one returns to the start node. The SOP and the PCATSP are almost equivalent [8] and a PCATSP instance can easily be reformulated as an SOP instance and vice versa.

![Figure 2.2: An example of a solution to a SOP instance (left) and a solution to the corresponding PCATSP instance (right). The numbers represent the order in which the nodes are visited.](image)

The biggest difference to the basic TSP that the SOP/PCATSP introduces is the addition of so-called precedence constraints. These constraints impose the rule that certain nodes are required to precede others (but not necessarily directly) in the solution. These problems are also asymmetric in the sense that the edges are directed and then called arcs. Because of the precedence constraints, the explicit order in which the nodes are visited is important and therefore the introduction of directed edges is necessary.

2.1.2 The Generalized Travelling Salesperson Problem

The GTSP is a variation of the TSP in which the set of nodes is partitioned into smaller sets. The problem is then to construct a minimum cost tour such that exactly one node in each set is visited. These sets are commonly referred to as clusters [9–13] but since this may indicate a configuration where the nodes within a set are closer to each other than to the nodes in other sets, and since this is not necessarily the case in the applications considered in this work, we will instead call these sets groups.
2.2. THE PRECEDENCE CONSTRAINED GENERALIZED TRAVELLING SALESPERSON PROBLEM

Figure 2.3: An example of a solution to a GTSP instance. The numbers represent the groups to which the nodes belong.

In a natural attempt to solve the GTSP two subproblems arise: group sequence and node choice, i.e. the order in which the groups are visited and the choice of the node that is to be visited in each group. The group sequence subproblem requires a fixed selection of which node that is to be visited within each group while the node selection subproblem requires a fixed order of the groups to be solved. It can be shown that solving the subproblems separately can result in the worst possible solution to a GTSP instance (see [11]). While there is a clear dependency between these subproblems, algorithms which separate or combine them to different degrees have, however, been shown to be efficient (see [12]).

2.2 The Precedence Constrained Generalized Travelling Salesperson Problem

The PCGTSP combines the PCATSP/SOP and the GTSP. It is a variation of the TSP where the node set is partitioned into groups and then precedence constraints are enforced on a group level, i.e. such that the groups are required to precede each other (but not necessarily directly). Since we are interested in modelling sequences of tasks (modelled as groups) where each task can be performed in different ways (modelled as nodes) it is natural to have the precedence constraints enforced on a group level as it is the tasks which are required to precede each other.

As with the GTSP, there are two subproblems which need to be solved. The group sequence subproblem, i.e. the problem of ordering the groups, which is affected by the precedence constraints, and the node selection subproblem which can be solved without taking the precedence constraints into account. An important note here is that since the PCGTSP is a generalization of an asymmetric GTSP we are mainly interested in research that pertains to the asymmetric GTSP.
Literature Review

While the PCGTSP has not been studied as much as the related problems SOP, PCATSP, and GTSP, many of the ideas and algorithms that have been developed for those problems can be useful when developing solution algorithms for the PCGTSP. To explore such a possibility, the previous research done on the asymmetric GTSP and SOP/PCATSP is reviewed in this chapter. Of particular interest is how the precedence constraints are handled in the case of SOP/PCATSP and how the two subproblems are solved for the case of the GTSP.

3.1 The Sequential Ordering Problem & The Precedence Constrained Asymmetric Travelling Salesperson Problem

Ascheuer et al. [7] have developed an integer linear programming model and a Branch-and-Cut algorithm for the PCATSP, capable of finding optimal solutions for large benchmark instances and instances from industrial applications within reasonable computation times. Many different valid inequalities and cutting planes were developed and implemented together with heuristic methods which provided upper bounds on the optimal value. The upper bounds found by the heuristics were found to be in need of improvement but were still good enough to help the Branch-and-Cut algorithm solve some of the larger problem instances.

Sarin et al. [14] have developed a polynomial length MILP formulation (i.e. the number of constraints and variables is of polynomial size with respect to the number of nodes) for the ATSP with and without precedence constraints. Different valid inequalities were tested using commercial software on small and medium sized problem instances with the goal of obtaining tight lower bounds.

Gambardella and Dorigo (1997) [8] have developed a state-of-the-art 3-opt local search heuristic together with an ant colony optimization (ACO) metaheuristic as a higher level framework for solving the SOP. The local search restricted the 3-exchanges to be path preserving, which made the verification of the precedence constraints computationally very efficient while still managing to improve the upper bound on the optimal objective function value considerably. The algorithm was able to solve many previously unsolved instances and produced tighter bounds on the (unknown) optimal values of several other instances. The algorithm has since then been improved by Gambardella et al. in 2012 [15].

Anghinolfi et al. [16] used the 3-opt local search heuristic developed by Gambardella and Dorigo [8] together with a discrete particle swarm optimization (PSO) metaheuristic for solving the SOP. Experimental results showed that the algorithm produced close to the same quality of upper bounds as the ACO algorithm presented in [8] for many of the problem instances tested.
3.2. THE GENERALIZED TRAVELLING SALESPERSON PROBLEM

Sung and Jeong [17] and Anthonissen [18] have developed genetic algorithms for solving the PCATSP/SOP. In [17] the precedence constraints were handled directly in each genetic operator by allowing only feasible solutions to be constructed. In [18] the precedence constraints were handled indirectly by adding a penalty term to the objective function for infeasible solutions. In both studies the results show that the quality of the upper bounds were significantly worse than the ones produced by the heuristic algorithms developed by Gambardella and Dorigo [8] and Anghinolfi [16].

3.2 The Generalized Travelling Salesperson Problem

Kara et al. [10] have formulated a polynomial length MILP model for the asymmetric GTSP. It is tested with commercial software to evaluate the lower bounds produced by the linear programming relaxation of the model. The lower bounds were shown to be better than other polynomial length formulations for the asymmetric GTSP.

Gutin et al. [13] developed an algorithm for the GTSP that combines many different local search methods together with a genetic algorithm metaheuristic. The algorithm outperformed the—at the time—state-of-the-art metaheuristic algorithm of Snyder and Daskin [19] with respect to quality of upper bounds in all of the standard benchmark problem instances [20] and found the global optimum for all instances but two. The algorithm developed but Gutin et al. [13] was however shown to be significantly slower than the one presented in [19].

Karapetyan and Gutin [11] developed an adaptation of the famous Lin-Kernighan \( k \)-opt heuristic [21] for the GTSP. In the experimental results it was shown that the Lin-Kernighan adaptation could compete with the best metaheuristic algorithms presented in [13] and [19] in terms of upper bounds on the optimal value.

Karatapetyan and Gutin [12] describe and analyse several local search heuristics for the GTSP which are used by Gutin et al. [13] and Karapetyan and Gutin [11]. Among them is an efficient dynamic programming algorithm which optimizes the selection of nodes given a fixed order of the groups. A procedure which locally improves the selection of nodes when performing different \( k \)-exchanges is also presented.

3.3 The Precedence Constrained Generalized Travelling Salesperson Problem

Castelino et al. [3] used a transformation which maps a PCGTSP to an SOP and then used a heuristic algorithm to solve the SOP. While there are no detailed results from their experiments they reported that the algorithm was ”...a viable option for the problem sizes normally encountered.”. However, the approach of transforming GTSPs to TSPs is not non-problematic. Solutions that are near-optimal in the TSP may be infeasible or very bad in the original GTSP (see [13]).

Dewil et al. [4] developed a heuristic framework for constructing and improving solutions to the laser cutting tool path problem. The problem is modelled as a PCGTSP but many application specific assumptions are imposed which makes the algorithm difficult to apply to general instances of the PCGTSP.
4

Integer Linear Programming

In this chapter we introduce some basic concepts and theoretical properties in integer linear programming (ILP) and mixed integer linear programming (MILP). Several relevant types of heuristic and optimizing algorithms will also be described. Any optimization problem from here on will assumed to be a minimization problem.

4.1 Linear Optimization

We formulate a general linear programming (LP) problem as:

\[ z^*_{LP} := \text{minimum } z(x) := c^T x \]

subject to \( Ax \leq b \)

\( x \in \mathbb{R}^n_+ \)

where \( z : \mathbb{R}^n \rightarrow \mathbb{R} \) denotes the objective function. We denote the set of feasible solutions by \( S_{LP} := \{ x \in \mathbb{R}^n_+ : Ax \leq b \} \). Note that it is possible to have equality constraints as well but these can be equivalently expressed as two inequality for each equality constraint. By replacing the constraint (4.1c) with \( x \in \mathbb{Z}^n_+ \) the optimization problem becomes an integer linear program (ILP) and we denote its set of feasible solutions by \( S_{ILP} := \{ x \in \mathbb{Z}^n_+ : Ax \leq b \} \). A general ILP problem:

\[ z^*_{ILP} := \text{minimum } z(x) := c^T x \]

subject to \( Ax \leq b \)

\( x \in \mathbb{Z}^n_+ \)

If an optimization problem includes both integer and non-integer variables it is known as a mixed integer linear programming (MILP) problem. Proceeding forward we will introduce some definitions and results regarding optimization theory, convexity, and computational complexity. This will help describe how ILP relates to LP and outlines the particular challenges of ILP problems. MILP and ILP problems face the same type of challenges as they both lose many of the properties that LP problems have when integer variables are introduced.
4.1 Computational complexity

To accurately describe and compare the difficulty and complexity of different problems and algorithms we must introduce some notation and concepts regarding computational complexity theory. The definitions and results presented here can be found in [5].

**Definition 4.1.1.** Let \( n \) be an appropriate measure of the length of the input data of a problem instance (e.g. the sizes of the matrices \( A, b \)). We say that an algorithm is a polynomial time algorithm if it takes \( O(f(n)) \) time to find a solution, where \( f \) is a polynomial function. We say that it is an exponential time algorithm if \( f \) is an exponential function.

**Definition 4.1.2.** Let \( \mathcal{P} \) be the class of problems which are solvable by a polynomial time algorithm.

**Definition 4.1.3.** Let \( \mathcal{NP} \) be the class of problems for which we are able to determine the feasibility of a problem instance by a nondeterministic polynomial time algorithm. For an ILP problem the feasibility of a problem instance is determined by checking if \((A, b) \in F\) where \( F = \{ (B, d) : \{ x \in \mathbb{Z}^n : Bx \leq d \} \neq \emptyset \} \).

Here, a nondeterministic polynomial algorithm is a general algorithm which consists of two phases. Phase one consists of guessing a solution to the current problem instance and phase two consists of checking whether the solution is feasible to the problem instance and giving an output in the case when it is feasible. An example of a nondeterministic polynomial time algorithm for a general ILP problem is given in [5, p. 129] as the following:

1. Guess an \( x \in \mathbb{Z}^n \).

2. If \( Ax \leq b \) then output \((A, b) \in F\); otherwise return.

Each call to this algorithm takes polynomial time but running it indefinitely until one receives an output does not necessarily take polynomial time.

**Definition 4.1.4.** A problem \( K \in \mathcal{NP} \) is defined as being \( \mathcal{NP} \)-complete if all problems in \( \mathcal{NP} \) can be reduced to \( K \) in polynomial time.

**Definition 4.1.5.** A problem \( K \) is \( \mathcal{NP} \)-hard if there is an \( \mathcal{NP} \)-complete problem which can be reduced to \( K \) in polynomial time.

\( \mathcal{NP} \)-complete problems are said to be the hardest problems in \( \mathcal{NP} \) and \( \mathcal{NP} \)-hard problems are at least as hard as \( \mathcal{NP} \)-complete problems. The next two results state that LP problems are solvable in polynomial time and that ILP problems are \( \mathcal{NP} \)-hard. For proofs see [5, Ch. I.6] and [5, Ch. I.5].

**Proposition 4.1.6.** Linear programming is in \( \mathcal{P} \).

**Proposition 4.1.7.** Integer linear programming is \( \mathcal{NP} \)-hard.

These results indicate that LP problems are significantly easier to solve than ILP problems. We continue by describing some of the properties that the polynomial time algorithms utilize when solving LP problems.

4.1.2 Convexity

Convexity is an important property, since convex optimization problems are considered solvable within reasonable time [22, p. 13]. As we will see from the definitions and the following results, LP problems are always convex. The results and definitions presented here can be found in [22].
Definition 4.1.8. A set \( S \subseteq \mathbb{R}^n \) is convex if for every two points \( x_1, x_2 \in S \) and every \( \lambda \in [0, 1] \) it holds that \( \lambda x_1 + (1 - \lambda) x_2 \in S \).

Definition 4.1.9. Let \( S \subseteq \mathbb{R}^n \) be a convex set. A function \( f : S \rightarrow \mathbb{R}^n \) is convex on the set \( S \) if for every two points \( x_1, x_2 \in S \) and every \( \lambda \in [0, 1] \) it holds that \( f(\lambda x_1 + (1 - \lambda) x_2) \leq \lambda f(x_1) + (1 - \lambda) f(x_2) \).

Definition 4.1.10. An optimization problem is convex if both its objective function and its feasible set are convex.

Proposition 4.1.11. An LP problem is convex.

The following proposition, together with Proposition 4.1.11, shows that for a convex optimization problem, and in particular an LP problem, finding a feasible solution \( x^* \) with neighbourhood \( N(x^*) = \{ x \in S_{LP} : z(x^*) \leq z(x), |x - x^*| < \epsilon, \epsilon > 0 \} \), i.e. a local optimum, is enough to find the global optimum. For a proof see [22, p. 76].

Proposition 4.1.12. For convex problems local optima are also global optima.

It is important to note that the set \( S_{ILP} \subseteq \mathbb{Z}^n \) is not convex. Hence, ILP problems do not have the property presented in Proposition 4.1.12 which follows from convexity.

4.1.3 Linearity

For LP problems the linearity of the constraints is also an important property, as demonstrated by Proposition 4.1.16 below. For a proof, see [5, p. 95]. The results and definitions presented below can be found in [5].

Definition 4.1.13. A polyhedron \( P \subseteq \mathbb{R}^n \) is a set of points that satisfy a finite number of linear inequalities. \( S_{LP} \) is a polyhedron.

Definition 4.1.14. A polyhedron \( P \subseteq \mathbb{R}^n \) is bounded if there exists a constant \( \omega \geq 0 \) such that the inclusion \( P \subseteq \{ x \in \mathbb{R}^n : |x_i| \leq \omega, i = 1, \ldots, n \} \) holds. A bounded polyhedron is called a polytope.

Definition 4.1.15. Let \( P \) be a polyhedron. We call \( x \in P \) an extreme point if there do not exist two points \( x_1, x_2 \in P \), where \( x_1 \neq x_2 \), and such that \( x = \frac{1}{2} x_1 + \frac{1}{2} x_2 \).

Proposition 4.1.16. If \( S_{LP} \) is a non-empty polytope and \( z^*_LP \) is finite then there is an optimal solution which is an extreme point to \( S_{LP} \).

One of the most widely used algorithms for solving LP problems, the simplex method, heavily exploits Proposition 4.1.16 by systematically searching among the extreme points to \( S_{LP} \). While the worst-case running time for the simplex method is exponential, in practice it often outperforms polynomial time algorithms [5, p. 122]. For a more in-depth description of the simplex algorithm we refer the reader to [5, pp. 33–40].

Since Proposition 4.1.16 does not hold for ILP problems, algorithms that—like the simplex method—rely on convexity of the problem to be solved cannot be directly used to solve ILP problems.

4.1.4 Relations between Linear Programming and Integer Linear Programming

We introduce some relations between LP and ILP and describe how these relations are utilized for solving ILP problems. Various algorithms which exploit these results will be described more thoroughly in Section 4.2.
4.2. OPTIMIZING ALGORITHMS

Definition 4.1.17. An LP relaxation of an ILP problem is the problem that results from removing the integrality constraints on the variables.

Definition 4.1.18. Let $X$ be a finite set of points in $\mathbb{R}^n$ and let $|X|$ denote the number of points in $X$. The convex hull of $X$ is the smallest convex set that contains $X$, and is then defined as $\text{conv}(X) := \left\{ \sum_{i=1}^{|X|} \lambda_i x_i : \sum_{i=1}^{|X|} \lambda_i = 1, \lambda_i \geq 0, i = 1, \ldots, |X| \right\}$.

Definition 4.1.19. Let $\text{ILP}_1$ and $\text{ILP}_2$ be two ILP formulations of the same problem, with optimal value $z^*_\text{ILP}_1$. Let $z^*_\text{ILP}_1$ and $z^*_\text{ILP}_2$ be the optimal values for the LP relaxations of $\text{ILP}_1$ and $\text{ILP}_2$ respectively. We say that $\text{ILP}_1$ is a stronger formulation than $\text{ILP}_2$ if it holds that $|z^*_\text{ILP}_1 - z^*_\text{ILP}_2| < |z^*_\text{ILP}_2 - z^*_\text{ILP}_2|$. We say that the formulation $\text{ILP}_1$ is ideal if it holds that $z^*_\text{ILP}_1 = z^*_\text{LP}$.

Definition 4.1.20. A valid inequality for an ILP problem is an affine constraint, as denoted by $\sum_{i=1}^n \pi_i x_i \leq \pi$, and where $\pi \in \mathbb{R}$ and $\pi_i \in \mathbb{R}$, $i = 1, \ldots, n$, are such that the constraint is fulfilled for all $x \in S_{\text{ILP}} = \{ y \in \mathbb{Z}^n_+ : Ay \leq b \}$.

Definition 4.1.21. $M_U \in \mathbb{R}$ is an upper bound on the value $z^*$ if it holds that $M_U \geq z^*$. $M_L \in \mathbb{R}$ is a lower bound on the value $z^*$ if it holds that $M_L \leq z^*$.

Assuming that $A$ in (4.1) is equal to $A$ in (4.2) and $b$ in (4.1) is equal to $b$ in (4.2), the LP problem (4.1) equals the LP relaxation of the ILP problem (4.2). Since LP problems are solvable in polynomial time, various algorithms utilize the solution to the LP relaxation to extract valuable information about the ILP problem. The following proposition follows from the fact that the inclusion $Z \subset R$ implies the inclusion $S_{\text{ILP}} \subseteq S_{\text{LP}}$.

Proposition 4.1.22. Let $z^*_\text{ILP}$ and $z^*_\text{LP}$ be the optimal values of an ILP problem and its LP relaxation respectively. Then $z^*_\text{LP}$ is a lower bound on $z^*_\text{ILP}$.

Proposition 4.1.22 is central to many algorithms for solving ILP problems. It states that the optimal value of the LP relaxation is always a lower bound on the optimal value of the original ILP problem. Similarly, for any $x \in S_{\text{ILP}}$, $z(x)$ is an upper bound on $z^*_\text{ILP}$. In the next section we will describe some optimizing algorithms which use these relations in the solution of ILP problems.

4.2 Optimizing Algorithms

Optimizing algorithms for ILP problems are exact methods which are mathematically proven to find the global optimum. They can be general and only rely on the fact that the problem can be formulated as an ILP problem (see [5, Ch. II.4]) or employ techniques based on the structure of a specific problem [5, p. 383]. However, as mentioned in Section 4.1.1, there does not exist any known polynomial time algorithm for general ILP problems. Because of this fact the optimizing algorithms designed for ILP problems can be computationally very demanding and can become impractical as the number of variables and constraints in a problem increases.

4.2.1 Cutting Plane Methods

Cutting plane algorithms try to approximate the convex hull of the set of feasible solutions by generating valid inequalities that “cut” away (discard) parts of the feasible set for the LP relaxed problem [5, pp. 349–351]. The general procedure can be described as follows.

1. Choose a strong ILP formulation for the problem.
2. Solve the corresponding LP relaxation.
3. If the solution is integer then it is optimal in the ILP problem and the procedure stops.

4. Generate valid inequalities that cut off the optimal solution for the LP relaxed problem by utilizing general or problem specific techniques.

5. Go to step 2.

In other words the goal is to introduce valid inequalities so that the ILP formulation becomes ideal. Cutting plane algorithms are seldom used by themselves since the number of cuts can become exponential. Instead cutting plane algorithms are often used to strengthen an ILP formulation so that the problem becomes easier to solve by another algorithm.

### 4.2.2 Branch-and-Bound

Branch-and-Bound algorithms use a strategy in which the set of feasible solutions is divided into subsets, which are then systematically searched. This strategy is generally known as divide and conquer, in which—if a problem is too hard to solve directly—the problem is solved by recursively dividing its set of feasible solutions into subsets and solving the problem over each separate subset [5, p. 352]. By using LP relaxations and the construction of feasible solutions, upper and lower bounds on the optimal value are generated which in turn help in discarding parts of the feasible set which contain suboptimal solutions.

The name Branch-and-Bound relates to how the division of the feasible set and the resulting subproblems can be visually represented by a tree made up by nodes and branches. Figure 4.1 illustrates a Branch-and-Bound tree resulting from the following problem instance:

$$
\begin{align*}
\text{minimize} & \quad z(x) := 7x_1 + 12x_2 + 5x_3 + 14x_4 \\
\text{subject to} & \quad -300x_1 - 600x_2 - 500x_3 - 1600x_4 \leq -700 \\
& \quad x_1, x_2, x_3, x_4 \in \{0, 1\}
\end{align*}
$$

![Branch-and-Bound search tree of an ILP problem with four binary variables](image)

Figure 4.1: Branch-and-Bound search tree of an ILP problem with four binary variables (see the problem instance (4.3)).
4.2. OPTIMIZING ALGORITHMS

At each node $i$ in the Branch-and-Bound search tree a subproblem—consisting of solving an LP relaxation, $\text{LP}_i$—is solved. So in the example above, at node 1 the LP relaxation of the original problem is solved, resulting in the solution $x = (0, 0, 0, 0.44)$. The node is then branched into two new nodes by rounding a non-integer valued variable up and down, respectively, to the nearest integers and keeping this value of the variable fixed. This is accomplished by adding new linear inequality constraints, resulting in new sets of feasible solutions for each new node. So node 1 is branched into node 2 and node 9 by adding the constraints $x_4 \leq 0$ and $x_4 \geq 1$, respectively. For the two new nodes the LP relaxation is solved again over their respective sets of feasible solutions. This branching procedure is then repeated until a feasible integer solution is found or a node in which the LP relaxation lacks a feasible solution is reached. In addition to this search procedure the algorithm keeps track of the best upper bound that is found, $M^*$, which corresponds to the objective function value of the best integer solution found. If a node $i$ has an LP relaxation for which it holds that $z^*_{\text{LP}_i} > M^*$, i.e. it is higher than the current best upper bound, then node $i$ is discarded and the algorithm ignores branching any further nodes from it. This procedure is known as pruning and is based on the fact that $z^*_{\text{LP}_i}$ provides a lower bound on $z(x)$, $x \in S_{\text{ILP}}$, for all nodes $j$ that might have resulted from the branching of node $i$ [5, p. 353]. In the example above $M^*$ is set to $M^* = 12$ when node 7 is searched. When node 8 is searched it is found that $z^*_{\text{LP}_8} = 13 > 12 = M^*$ so node 8 is pruned. The solution found in node 9 is also discarded since it is found to be worse than the one found in node 7.

There are many different strategies for choosing which nodes to search first and which variables to branch [5, pp. 358–359]. Their effectiveness is mostly dependent on the structure of the problem and what the user aims to achieve with the algorithm. For example one could choose what is called a depth-first search, in which the Branch-and-Bound algorithm always favours either the left or the right branch. This puts emphasis on searching the deepest possible nodes first. The benefit of this strategy is that the algorithm finds a feasible integer solution faster as these are often found deep in the search tree [5, p. 358].

4.2.3 Branch-and-Cut

Branch-and-Cut algorithms are types of Branch-and-Bound algorithms in which cutting plane methods are incorporated to generate valid inequalities for the ILP problem [5, pp. 388–392]. By strengthening the LP relaxation of the problem the bounding part of the algorithm is provided with tighter lower bounds which enable the algorithm to prune more nodes at earlier stages.

4.2.4 Dynamic Programming

By utilizing special structures of an optimization problem a dynamic programming algorithm decomposes the problem into nested subproblems which are then solved using a recursive strategy [5, p. 417]. As an example, the problem of finding the shortest path in a directed acyclic graph is a typical problem for which a dynamic programming algorithm is well-suited to solve [5, p. 419].

The subproblems are sequentially solved given some initial state. A state in this context is a set of conditions under which the next subproblem must be solved. Solving a subproblem produces a new state which will then affect the solution of the subsequent subproblems [5, p. 417]. A specific example of a dynamic programming algorithm is presented in Section 6.2.

The difficulty when using dynamic programming is often to identify the division of the problem into subproblems and their nested structure. However, for the problem studied in this thesis the dynamic programming approach will be utilized when solving shortest path problems, so we are not faced with such difficulties.

14
4.3 Heuristic Algorithms

While the optimizing methods are guaranteed to find the optimal value for an optimization problem (although, sometimes, only after a very long computation time), heuristic algorithms sacrifice the guarantee of optimality and focus on approximating the optimal value while retaining a reasonably short computing time. They range from being problem specific algorithms to being general solution strategies that apply to many different classes of problems [5, p. 393].

Since ILP problems are \( \mathcal{NP} \)-hard, heuristic algorithms are often employed as complements to optimizing algorithms but are also used independently when optimizing algorithms are deemed to be too slow or when a suboptimal solution quality is acceptable [5, p. 393].

4.3.1 Construction Heuristics

Construction heuristics are methods whose aim is to create a feasible solution from scratch. Using some intuitive rule built around knowledge about the optimization problem’s structure the algorithm creates a feasible solution.

An example of a construction heuristic for the TSP is the nearest neighbour heuristic [5, pp. 475–476]. It creates a solution by randomly choosing a starting node and then iteratively adding minimum cost arcs—which connect to nodes not yet visited—to the solution. When all nodes have been visited a final arc from the last to the first node is added. Solutions generated by this algorithm are guaranteed to be feasible, given that the graph is complete, but cannot be guaranteed to be optimal [5, p. 476].

4.3.2 Local Search Heuristics

Local search heuristics are a class of algorithms that given an initial feasible solution try to find an improved feasible solution by applying some form of change to it [5, p. 394]. This change is performed iteratively to the resulting solutions until no further improvement can be achieved by the specific change. Such a solution is referred to as locally optimal [5, p. 407].

Another way to describe local search heuristics is to think of them as defining neighbourhoods around a given feasible solution. The local search method defines a set of feasible solutions \( N(x) \subseteq S_{ILP} \)—or a neighbourhood—around a feasible solution \( x \in S_{ILP} \). The solutions in \( N(x) \) are such that they can be reached by some heuristic rule of change to \( x \). So, starting from an initial solution, \( x_0 \), a local search heuristic then iteratively chooses an improved feasible solution \( x_{k+1} \in N(x_k) \) until it arrives at a locally optimal solution. A solution, \( x^*_{LO} \in S_{ILP} \), is defined as locally optimal if \( z(x^*_{LO}) \leq z(x) \) for all \( x \in N(x^*_{LO}) \). Local search heuristics can employ different strategies for choosing an improved solution in a given neighbourhood. They can either choose any feasible and improving solution \( x_{k+1} \in N(x_k) \) such that \( z(x_{k+1}) \leq z(x_k) \) or they can choose the best solution \( x_{k+1} \in N(x_k) \) such that \( z(x_{k+1}) \leq z(y) \), \( \forall y \in N(x_k) \).

There are a multitude of local search heuristics developed for the TSP and the GTSP but these are often ineffective when precedence constraints are introduced. Therefore modifications of existing (or new) methods must be developed and evaluated for the PCGTSP.

The \( k \)-opt Local Search

The \( k \)-opt local search is an effective local search method which is widely used for the TSP and TSP-like problems. The idea for a \( k \)-opt algorithm is to remove \( k \) arcs from a feasible tour and replace them by \( k \) other arcs in such a way that the resulting solution is feasible and
improved. This procedure is known as a \( k \)-exchange. The computational complexity of finding the best tour in a series of \( k \)-opt neighbourhoods—a so-called \( k \)-optimal tour—has been shown to be \( O(n^k) \), where \( n \) denotes the number of nodes in a problem instance [8]. A \( k \)-optimal tour, \( x^*_{k\text{-opt}} \) is a feasible solution such that \( z(x^*_{k\text{-opt}}) < z(y) \) for all \( y \in N_{k\text{-opt}}(x) \), where \( x \) is a feasible solution. Usually only algorithms with \( k \leq 5 \) are considered, since the \( k \)-exchange procedure is typically repeated many times for each problem instance.

The Lin-Kernighan algorithm [21], which is one of the most effective heuristic algorithms designed for the TSP, is built around the \( k \)-opt local search. As mentioned in Chapter 3, Karapetyan et al. [11] have formulated a generalization of the Lin-Kernighan heuristic for the GTSP with very good results. For precedence constrained problems, however, the search for feasible \( k \)-exchanges can become computationally very cumbersome, since it is required to verify that the precedence constraints are satisfied. Therefore, specially adapted \( k \)-opt algorithms that can verify the precedence constraints in an efficient way are required.

4.3.3 Metaheuristics

Metaheuristic methods iteratively generate solutions to an optimization problem, and through a process of identifying characteristics of good solutions they replicate those in the solutions in subsequent iterations. While the manner in which the solutions are generated differs between different types of algorithms, the main goal of all metaheuristics is to explore diverse parts of \( S_{\text{ILP}} \) and avoid getting stuck at local optima (see [23]).

The principle of generating solutions and the replication of good solutions is known as diversification and intensification, respectively, of the set of feasible solutions [23]. Diversification occurs when the metaheuristic tries to generate diverse feasible solutions to the problem and intensification occurs when the metaheuristic tries to explore solutions similar to those that are deemed to be good according to some measure of fitness. The fitness value can vary depending on the specific problem to be solved but for ILP problems this is often just taken to be \( z(x) \).

Hybrid Metaheuristics

Metaheuristics are generally not problem-specific, which make them suitable to use as a higher level framework for local search heuristics. Such algorithms which combine a metaheuristic and local search heuristics are called hybrid metaheuristics ([23]). Through the diversification and intensification property of the metaheuristic the local search method is guided to a diverse set of initial solutions which allows the local search to find many locally optimal solutions and thus greatly increases the chances of finding a good feasible solution.
5

Mathematical Modelling

In this chapter we propose a polynomial size MILP model for the precedence constrained generalized travelling salesperson problem (PCGTSP) as presented in Section 2.2. The MILP model for the PCGTSP is based on the model of Kara et al. ([10]) for the generalized travelling salesperson problem (GTSP) (hereby referred to as GTSP1) and the model of Sarin et al. ([14]) for the precedence constrained asymmetric travelling salesperson problem (PCATSP) (hereby referred to as PCATSP1). We begin by introducing some general notation and defining the basic models PCATSP1 and GTSP1.

5.1 Notation

Let \( n \) be the number of nodes in a problem instance and let \( V := \{1, \ldots, n\} \) denote the set of all nodes. Let \( A := \{(i, j) : i, j \in V, i \neq j\} \) denote the set of all (directed) arcs between all nodes and let \( c_{ij}, i, j \in V \), denote the cost associated with the arc from node \( i \) to node \( j \). As described in Section 2.1.2, the node set, \( V \), in the GTSP and the PCGTSP is partitioned into groups. Therefore, let \( G := \{1, \ldots, m\} \) denote the set of all group indices and let \( V_1, \ldots, V_m \) be a partition of \( V \) where \( V_p, p \in G \), is called a group. The partition of \( V \) must satisfy the constraints \( V_p \neq \emptyset \), \( V = \bigcup_{p \in G} V_p \) and \( V_p \cap V_q = \emptyset \) when \( p \neq q \). As mentioned in Section 2.1.1 the PCATSP and the PCGTSP are affected by precedence constraints which are defined by sets. For the PCATSP the precedence constraint sets are denoted as \( \text{PN}_j := \{i \in V : \text{node } i \text{ must precede node } j \text{ in the tour}\} \), \( j \in V \), since the precedence constraints are enforced on the nodes. For the PCGTSP the precedence constraint sets are denoted as \( \text{PG}_q := \{p \in G : \text{group } p \text{ must precede group } q \text{ in the tour}\} \), \( q \in G \), since the precedence constraints are enforced on the groups.

We say that a formulation is of polynomial length if the number of constraints and variables is \( \mathcal{O}(f(n, m)) \), where \( f \) is a polynomial function.

5.2 The Model GTSP1

In addition to the notation presented in Section 5.1 the model of Kara et al. [10] uses the following variables:
5.3. THE MODEL PCATSP1

\[ x_{ij} := \begin{cases} 1, & \text{if the traveller moves directly from node } i \text{ to } j, \\ 0, & \text{otherwise,} \end{cases} \quad i, j \in V, \]

\[ w_{pq} := \begin{cases} 1 & \text{if the traveller moves directly from group } p \text{ to } q, \\ 0 & \text{otherwise,} \end{cases} \quad p, q \in G, \]

\[ u_p := \text{the position of group } p \text{ in the tour} \quad p \in G. \]

The GTSP1 model is formulated as

\[
\text{minimize } z(x) := \sum_{i \in V} \sum_{j \in V \setminus \{i\}} c_{ij} x_{ij}, \quad (5.1a)
\]

subject to

\[
\sum_{i \in V_p} \sum_{j \in V \setminus V_p} x_{ij} = 1, \quad p \in G, \quad (5.1b)
\]

\[
\sum_{i \in V \setminus V_p} \sum_{j \in V_p} x_{ij} = 1, \quad p \in G, \quad (5.1c)
\]

\[
\sum_{j \in V \setminus \{i\}} (x_{ji} - x_{ij}) = 0, \quad i \in V, \quad (5.1d)
\]

\[
\sum_{i \in V_p} \sum_{j \in V_q} x_{ij} = w_{pq}, \quad p, q \in G : p \neq q, \quad (5.1e)
\]

\[
u_p - u_q + (m - 1)w_{pq} + (m - 3)w_{qp} \leq m - 2, \quad p, q \in G \setminus \{1\} : p \neq q, \quad (5.1f)\]

\[
u_p - \sum_{q \in \{2, \ldots, k\} \setminus \{p\}} w_{qp} \geq 1, \quad p \in G \setminus \{1\}, \quad (5.1g)
\]

\[
u_p + (m - 2)w_{1p} \leq k - 1, \quad p \in G \setminus \{1\}, \quad (5.1h)
\]

\[
u_p \geq 0, \quad p \in G, \quad (5.1i)
\]

\[
w_{pq} \geq 0, \quad p, q \in G : p \neq q, \quad (5.1j)
\]

\[
x_{ij} \in \{0, 1\}, \quad i, j \in V : i \neq j. \quad (5.1k)
\]

The objective function (5.1a) sums up the costs associated with each activated arc. The constraints (5.1c) and [(5.1b)] ensure that every group of nodes is entered [exited] exactly once from [to] a node belonging to another group and the constraints (5.1d) ensure that if a node is entered it must also be exited. The constraints (5.1e) together with (5.1b)–(5.1d) guarantee that the variable \( w_{pq} = 1 \) when an arc from group \( p \) to group \( q \) is traversed. The constraints (5.1f) are a so-called subtour elimination constraints and ensure that no closed subtours are allowed in a solution. The constraints (5.1g)–(5.1h) bound and initialize the variables \( u_p \) such that if \( w_{pq} = 1 \) then the constraint \( u_p = u_q + 1 \) must hold. The model GTSP1 has \( n^2 + m^2 + m \) variables and \( n^2 + n + 3m^2 + 5m \) constraints.

5.3 The Model PCATSP1

In the model of Sarin et al. [14] the variables
\[ x_{ij} := \begin{cases} 1 & \text{if the traveller moves directly from node } i \text{ to } j, \\ 0 & \text{otherwise,} \end{cases} \quad i, j \in V, \]

\[ y_{ij} := \begin{cases} 1 & \text{if node } i \text{ precedes node } j, \text{ but not necessarily directly, in the tour,} \\ 0 & \text{otherwise,} \end{cases} \quad i, j \in V, \]

are used. The PCATSP1 model is formulated as

\[
\begin{align*}
\text{minimize } z(x) := & \sum_{i \in V} \sum_{j \in V \setminus \{i\}} c_{ij} x_{ij}, \\
\text{subject to } & \sum_{i \in V \setminus \{j\}} x_{ij} = 1, \quad j \in V, \quad (5.2b) \\
& \sum_{i \in V \setminus \{j\}} x_{ij} = 1, \quad i \in V, \quad (5.2c) \\
& y_{ij} - x_{ij} \geq 0, \quad i, j \in V : i \neq j, \quad (5.2e) \\
& y_{ij} + y_{ji} = 1, \quad i, j \in V : i \neq j, \quad (5.2f) \\
& (y_{ij} + x_{ji}) + y_{jr} + y_{ri} \leq 2, \quad i, j, r \in V : i \neq j \neq r \neq i, \quad (5.2g) \\
& y_{ij} = 1, \quad i \in PN_j, j \in V \quad (5.2h) \\
& y_{ij} \geq 0, \quad i, j \in V : i \neq j, \quad (5.2i) \\
& x_{ij} \in \{0,1\}, \quad i, j \in V : i \neq j. \quad (5.2j)
\end{align*}
\]

The objective function (5.2a) is equivalent to (5.1a). The constraints (5.2c) and (5.2d) ensure that the traveller enters and leaves each node exactly once. The constraints (5.2e) guarantee that the variables \( y_{ij} \) are non-zero whenever node \( j \) is visited after node \( i \). The constraints (5.2f) ensure that if node \( i \) precedes node \( j \) then node \( j \) cannot precede node \( i \), and together with the constraints (5.2e) they imply a binary constraint on the variables \( y_{ij} \) provided that \( x_{ij} \) are binary valued. The constraints (5.2f) constitute the subtour elimination constraints of the model. The constraints (5.2h) enforce the precedence constraints defined by the sets \( PN_j, j \in V \) by forcing \( y_{ij} = 1 \) if node \( i \) must precede node \( j \) in a feasible solution. For a detailed proof of the validity of the formulation 5.2 see [14]. The model PCATSP1 has \( 2n^2 \) variables and \( n^3 + 5n^2 + 2n \) constraints.

5.4 A proposed PCGTSP model

For the proposed PCGTSP model we combine the GTSP1 and PCATSP1 (i.e. (5.1) and (5.2), respectively) models with some modifications. We use the subtour elimination constraints (5.2g) as defined in the PCATSP1 and hence eliminate the need for the group order variables \( u_p \). Furthermore, since the precedence constraints are enforced on a group level, the variables \( y_{ij} \) are replaced by the corresponding variables \( v_{pq} \) which depend on the group set \( G \) instead of the node set \( V \). Naturally the precedence constraint sets \( PN_j \) are also changed to depend on the groups. So they are expressed as \( PG_q := \{ p \in G : \text{group } p \text{ must precede group } q \text{ in the tour} \} \).

The variables used in the PCGTSP model are the following:

\[
\text{PG}_q := \{ p \in G : \text{group } p \text{ must precede group } q \text{ in the tour} \}.
\]
5.4. A PROPOSED PCGTSP MODEL

\[ x_{ij} := \begin{cases} 
1 & \text{if the traveller moves directly from node } i \text{ to } j, \\
0 & \text{otherwise,} 
\end{cases} \quad i, j \in V, \]

\[ v_{pq} := \begin{cases} 
1 & \text{if group } p \text{ precedes group } q, \text{ but not necessarily directly, in the tour,} \\
0 & \text{otherwise,} 
\end{cases} \quad p, q \in G, \]

\[ w_{pq} := \begin{cases} 
1 & \text{if the traveller moves directly from group } p \text{ to } q, \\
0 & \text{otherwise,} 
\end{cases} \quad p, q \in G. \]

The PCGTSP model is formulated as

\[
\text{minimize } z(x) := \sum_{i \in V} \sum_{j \in V \setminus \{i\}} c_{ij} x_{ij}, \quad (5.3a)
\]

subject to

\[
\sum_{i \in V} \sum_{j \in V \setminus \{i\}} x_{ij} = 1, \quad p \in G, \quad (5.3b)
\]

\[
\sum_{i \in V \setminus \{i\}} \sum_{j \in V \setminus \{i\}} x_{ij} = 1, \quad p \in G, \quad (5.3c)
\]

\[
\sum_{j \in V \setminus \{i\}} (x_{ji} - x_{ij}) = 0, \quad i \in V, \quad (5.3d)
\]

\[
\sum_{i \in V \setminus \{i\}} \sum_{j \in V \setminus \{i\}} x_{ij} = 0, \quad p \in G, \quad (5.3e)
\]

\[
\sum_{i \in V \setminus \{i\}} \sum_{j \in V_q} x_{ij} = w_{pq}, \quad p, q \in G : p \neq q, \quad (5.3f)
\]

\[
v_{pq} - w_{pq} \geq 0, \quad p, q \in G : p \neq q, \quad (5.3g)
\]

\[
v_{pq} + v_{qp} = 1, \quad p, q \in G : p \neq q, \quad (5.3h)
\]

\[
(v_{pq} + w_{qp}) + v_{qr} + v_{rp} \leq 2, \quad p, q, r \in G : p \neq q \neq r \neq p, \quad (5.3i)
\]

\[
v_{pq} = 1, \quad p \in PG_q, \quad q \in G, \quad (5.3j)
\]

\[
v_{pq} \geq 0, \quad p, q \in G : p \neq q, \quad (5.3k)
\]

\[
w_{pq} \geq 0, \quad p, q \in G : p \neq q, \quad (5.3l)
\]

\[
x_{ij} \in \{0, 1\}, \quad i, j \in V : i \neq j. \quad (5.3m)
\]

By removing the variables \(u_p\), the constraints (5.1f)–(5.1h) in the model GTSP1 can also be eliminated. The variables \(w_{pq}\) will be sufficiently bounded by the constraints (5.3f). Another important difference between the model GTSP1 and the proposed PCGTSP model is the expression of the constraints (5.2e). In the model PCATSP1 (i.e. (5.2)) the variables \(y_{ij}\) are bounded by the variables \(x_{ij}\) since the precedence constraints are enforced on the nodes in the PCATSP. In the proposed PCGTSP model the corresponding group precedence variables \(v_{pq}\) are bounded by the variables \(w_{pq}\) because of the group dependency. Another addition to the constraints of the model GTSP1 are (5.3e) which prevent moves between nodes within any of the groups. These constraints are needed since we do not want make any assumptions about the arc costs for the applications considered in this thesis. Therefore, arcs within each group are explicitly disallowed in a feasible solution by the constraints (5.3e). The proposed PCGTSP model has \(n^2 + 2m^2\) variables and \(n^2 + n + m^3 + 6m^2 + 3m\) constraints.
6

Algorithms

We next describe the specific algorithms that have been implemented and evaluated for the PCGTSP in this thesis.

6.1 Local methods

Local search heuristics have been used for the TSP and its variations with great success in many cases (see e.g. [6, ch. 5–10] and [12, 21, 24]). Therefore, a natural approach for developing algorithms for the PCGTSP is to adapt and generalize some of the local search heuristics developed for the TSP to the PCGTSP.

The most successful heuristic algorithms for the TSP are built around the $k$-opt local search (see e.g. [24]). We continue by describing some $k$-opt algorithms adapted to handle the precedence constraints and to take the node selection into account in order to more effectively find good solutions for the PCGTSP.

6.1.1 Lexicographic Path Preserving 3-opt

Gambardella and Dorigo ([8]) have developed a 3-opt local search heuristic, specifically for fast verification of the precedence constraints appearing in the SOP/PCATSP. By simply categorizing the precedence constraints as acting on all the nodes within each group the method can be applied to the PCGTSP.

When removing three arcs from a PCATSP/SOP tour, their replacement by three new arcs can be done in four different ways. One is path preserving, i.e. it preserves the orientation of the tour, and three are path inverting, i.e. they invert the orientation of one or several segments of the tour. Figure 6.1 is an illustration of the four possible 3-exchanges. The arcs between the nodes $t_1$ and $t_2$, $t_3$ and $t_4$, and $t_5$ and $t_6$ are removed and then replaced by three other arcs. For the path preserving 3-exchange (A) the clockwise orientation of the tour is preserved throughout all segments while for the path-inverting 3-exchanges (B), (C) and (D), some of the tour segments are visited in a counter-clockwise direction. The lexicographic path preserving 3-opt (LPP 3-opt) excludes the path inverting 3-exchanges and considers only the path preserving one. The reason for only considering path preserving 3-exchanges is that it enables the algorithm to verify the feasibility of the proposed 3-exchange and the 3-optimality of the tour in $O(n^3)$ time [8]. In addition to this, for asymmetric problems the improvement in the objective function after a path preserving 3-exchange can be calculated in $O(1)$ time, while for the path inverting 3-exchanges the analogous computation has a worst case complexity of $O(n)$. 
6.1. LOCAL METHODS

The LPP 3-opt algorithm takes a feasible tour as input and performs a search in which it identifies two subsequences, path-left and path-right. The algorithm then checks if swapping the two subsequences, i.e. interchanging places, results in a feasible and improved tour by confirming that the precedence constraints are still satisfied after the swap and that the total cost of the tour is decreased by executing the 3-exchange. For the 3-exchange to be feasible the subsequences path-left and path-right must be defined in a way such that none of the nodes in path-left are required to precede the nodes in path-right.

Let $T = \{t_1, \ldots, t_m\}$ be a feasible tour. The LPP 3-opt then performs a forward search by initializing three indices $h, i, j$, which point to nodes in the ordered set $T$, as $h = 1$, $i = h + 1$ and $j = i + 1$ and then identifying the two consecutive subsequences as path-left := \{t_{h+1}, \ldots, t_i\} and path-right := \{t_{i+1}, \ldots, t_j\}. The algorithm then expands path-right by incrementally increasing $j$ by one until it points to $t_m$ or a node $t_p$ which must succeed some node $t_q \in$ path-left. When the loop over $j$ is terminated, $i$ is increased by one, $j$ is reinitialized as $j = i + 1$ and path-right is expanded again. When path-left has been expanded to the point where $i = m - 1$, $h$ is increased by one and the whole process is repeated. For every fixed $h$ the LPP 3-opt performs a forward and a backward search. For the backward search the indices $i$ and $j$ are initialized as $i = h - 1$ and $j = i - 1$, the subsequences are identified as path-left := \{t_{j+1}, \ldots, t_i\} and path-right := \{t_{i+1}, \ldots, t_h\} and the subsequences are expanded by incrementally decreasing the indices $i$ and $j$. The LPP 3-opt continues to search and execute improving 3-exchanges until a tour is found to be locally optimal. A locally optimal solution in this context is a feasible solution $x_{LO}^*$ which cannot be improved by performing a path preserving 3-exchange.

![Figure 6.1: Four possible 3-exchanges. (A) is path preserving while (B), (C), and (D) are path inverting.](image1)

![Figure 6.2: Identification of path-left and path-right during a forward search before the 3-exchange.](image2)

![Figure 6.3: Identification of path-left and path-right during a forward search after the 3-exchange.](image3)
6.1. LOCAL METHODS

To make sure that the precedence constraints are satisfied the LPP 3-opt algorithm performs a labelling procedure in which each node \( t_p \in T \) is labelled as feasible to add to the subsequences. In the case of the forward search, each time \( \text{path-left} \) is expanded by the addition of a new node \( t_p \), every node \( t_q \) such that \( q > p \), and which must succeed \( t_p \) is labelled as infeasible to add to \( \text{path-right} \). Analogously, in the case of backward search each time \( \text{path-right} \) is expanded by the addition of a new node \( t_p \), every node \( t_q \in T, q < p \) that must precede \( t_p \) is labelled as infeasible to add to \( \text{path-left} \).

While the goal of a \( k \)-opt algorithm is generally to find a \( k \)-optimal tour, the search for a \( k \)-optimal solution has been found to be too time consuming while the gain in solution quality is typically very small [8, 12]. Instead Gambardella and Dorigo [8] found that terminating the search within the loop which expands \( \text{path-left} \) in the case of a forward search and the loop which expands \( \text{path-right} \) in the case of a backward search as soon as a feasible and improving 3-exchange was found gave the best results for the LPP 3-opt algorithm. When applying the LPP 3-opt to the PCGTSP we will use this search strategy.

Furthermore, Gambardella and Dorigo [8] present various ways of choosing \( h \). Since the expansion of \( \text{path-left} \) and \( \text{path-right} \) can be done independently of the choice of \( h \) the algorithm is not constrained to choosing \( h \) in a strict increasing sequence. So instead of choosing \( h = 1, 2, \ldots, m-2 \) in that order one can choose the values on \( h \) in some permuted order instead. The strategy which was shown to give the best results was the “don’t push stack” strategy. In the beginning of the LPP 3-opt a stack is initialized with all possible values (i.e. \( 1, \ldots, m \)) for \( h \). The value for the index \( h \) is then popped off the top of the stack each time \( h \) would have been increased. When a feasible 3-exchange is executed the nodes that are involved in the 3-exchange (i.e. \( h, i, j, h+1, i+1, \) and \( j+1 \)) are pushed onto the top of the stack if they do not already belong to the stack. So each time a feasible 3-exchange is executed with \( h = p \) and the whole search for a new feasible exchange starts over, the value of \( h \) will remain equal to \( p \). This strategy is also used when implementing the LPP 3-opt for the PCGTSP.
Algorithm 6.1 Lexicographic Path Preserving 3-opt

1. Let $T = \{t_1, \ldots, t_m\}$ be the initial tour given to the LPP 3-opt. Initialize $improvement := true$

2. If $improvement := true$, then initialize the stack $S = \{1, \ldots, m\}$. Otherwise return $T$ and terminate the algorithm.

3. Set $improvement := false$.

4. Pop off the top value of $S$ and let $h$ be equal to that top value. So $h := 1$ and $S = \{2, \ldots, m\}$.

5. Initialize $count_h := 1$, and let $fMark = [0, \ldots, 0]$ and $bMark = [0, \ldots, 0]$ be vectors with $n$ elements.


7. Set $i_F := h + 1$ and $j_F := i_F + 1$.

8. If $bestExchange$ is empty, $i_F < m$, and $i_F < j_F$, then set $fMark_{i_F} := count_h$ for all $t_p \in (t_{p+1}, t_{p+2}, \ldots, t_m)$ such that $t_p$ must precede $t_i$ (as dictated by the precedence constraints defined in the sets $G_q$, $q \in G$). Otherwise go to step 12.

9. If $j_F \leq m$ and $fMark_{t_j} \neq count_h$, then compute $C_{old} := c_{t_i, t_{b+1}} + c_{t_{b+1}, t_{b+2}} + c_{t_{b+2}, t_{b+3}} + \ldots + c_{t_m, t_1}$ and $C_{new} := c_{t_i, t_{b+1}} + c_{t_{b+1}, t_{b+2}} + \ldots + c_{t_m, t_1}$. Otherwise set $i_F := i_F + 1$ and go to step 8.

10. If $C_{new} < C_{old}$ and $C_{new} < C^*$, then set $bestExchange := \{h, i_F, j_F\}$, forward := true, backward := false, and $C^* := C_{new}$.

11. Set $j_F := j_F + 1$ and go to step 9.

12. Set $i_B := h - 1$ and $j_B := i_B - 1$.

13. If $bestExchange$ is empty, $i_B > 1$, and $i_B > j_B$, then set $bMark_{i_B} := count_h$ for all $t_p \in (t_1, t_2, \ldots, t_{i_B})$ such that $t_p$ must precede $t_{b+1}$ (as dictated by the precedence constraints defined in the sets $G_q$, $q \in G$). Otherwise go to step 17.

14. If $j_B \geq 0$ and $bMark_{t_{b+1}} \neq count_h$, then compute $C_{old} := c_{t_{i_B}, t_{b+1}} + c_{t_{b+1}, t_{b+2}} + c_{t_{b+2}, t_{b+3}} + \ldots + c_{t_m, t_1}$ and $C_{new} := c_{t_{i_B}, t_{b+1}} + c_{t_{b+1}, t_{b+2}} + \ldots + c_{t_m, t_1}$. Otherwise set $i_B := i_B - 1$ and go to step 13.

15. If $C_{new} < C_{old}$ and $C_{new} < C^*$, then set $bestExchange := \{h, i_B, j_B\}$, backward := true, forward := false, and $C^* := C_{new}$.

16. Set $j_B := j_B - 1$ and go to step 14.

17. If $bestExchange$ is not empty and forward = true, then let $h^* := bestExchange_1$, $i^* := bestExchange_2$, and $j^* := bestExchange_3$, set $\tilde{T} := \{\tilde{t}_1, \ldots, \tilde{t}_m\} := \{t_1, t_2, \ldots, t_{i_B}, t_{i_B+1}, \ldots, t_{i^*-1}, t_{i^*+1}, \ldots, t_{j^*-1}, t_{j^*+1}, \ldots, t_m\}$, set $T := \tilde{T}$, set $improvement := true$, push the values $h^*, i^*, j^*, h^* + 1, i^* + 1, j^* + 1$ onto $S$ if they do not already belong to it, and go to step 19.

18. If $bestExchange$ is not empty and backward = true, then let $h^* := bestExchange_1$, $i^* := bestExchange_2$, and $j^* := bestExchange_3$, set $\tilde{T} := \{\tilde{t}_1, \ldots, \tilde{t}_m\} := \{t_1, t_2, \ldots, t_{j_B}, t_{j_B+1}, \ldots, t_{j^*-1}, t_{j^*+1}, \ldots, t_m\}$, set $T := \tilde{T}$, set $improvement := true$, and push the values $h^*, i^*, j^*, h^* + 1, i^* + 1, j^* + 1$ onto $S$ if they do not already belong to it.

19. If $S$ is not empty then pop off the top value of $S$ and let $h$ be equal to that top value, set $count_h := count_h + 1$, and go to step 7. If $S$ is empty then go to step 2.
6.1.2 Double Bridge

The LPP 3-opt algorithm can be generalized for other path preserving k-exchanges for precedence constrained problems. One often used path preserving k-exchange is the so called Double Bridge (DB) move, which is a special case of a 4-exchange (see [6, pp. 327, 416, 462–463]).

The LPP 3-opt algorithm can be adapted to find feasible DB moves by only extending it slightly. Performing the DB move consists of identifying three consecutive subsequences \((\text{path}_1, \text{path}_2, \text{path}_3)\) within a feasible tour and then permuting them such that their new order becomes \((\text{path}_3, \text{path}_2, \text{path}_1)\). To achieve this within the framework of the LPP 3-opt algorithm a new index \(k\) is introduced. So the indices which are used by the DB algorithm are \(h, i, j, k\) and in the case of a forward search the subsequences are identified as \(\text{path}_1 := \{t_{h+1}, \ldots, t_i\}\), \(\text{path}_2 := \{t_{i+1}, \ldots, t_j\}\) and \(\text{path}_3 := \{t_{j+1}, \ldots, t_k\}\).

Other than these changes the rest of the algorithm remains largely the same. For each \(h\) a backward and forward search is performed and every time a subsequence is expanded by increasing the indices a labelling procedure marks nodes as infeasible if they are to succeed or precede the new node. As with the LPP 3-opt algorithm, the "don't push stack" strategy is used and the search is also terminated within the second inner-most loop (in this case the loop which expands \(\text{path}_2\)) if a feasible and improving exchange has been found. As a result of these extensions the worst case computing time for the DB algorithm algorithm is \(O(m^4)\).

6.1.3 Path Inverting 2-opt

While it is possible to perform path preserving 3-exchanges, 2-exchanges are always path inverting. Even though path inverting k-exchanges are theoretically inefficient compared to path preserving ones, when applied to asymmetric and precedence constrained problems, it may be interesting to evaluate path inverting k-exchanges with respect to the possible improvement of PCGTSP tours.
Let $T = \{t_1, \ldots, t_m\}$ be a feasible tour. The Path Inverting (PI) 2-opt identifies a subsequence within a tour $T$ and checks whether the precedence constraints are still satisfied when this subsequence is inverted. This is achieved by creating a nested loop, in which two indices, $i$ and $j$, are incrementally increased. Each time $j$ is increased a check is performed to see whether the given tour remains feasible when the subsequence $\{t_i, \ldots, t_j\}$ is inverted. Every time $i$ is increased $j$ is set to $j = i + 1$ and the search starts over. As with the LPP 3-opt and the DB algorithms, the PI 2-opt does not search for the 2-optimal tour but instead executes a 2-exchange which is feasible and improving as soon as it is found. The worst case computing time of the PI 2-opt is $O(m^3)$.

**Algorithm 6.2 Path Inverting 2-opt**

1. Let $T = \{t_1, \ldots, t_m\}$ be the initial tour given to the LPP 3-opt with cost $C_T$. Initialize $\text{improvement} := true$.

2. If $\text{improvement} := true$, then set $\text{improvement} := false$. Otherwise return $T$ and terminate the algorithm.


4. If $\text{improvement} = false$ and $i < m$, then set $\text{stopExpand} := false$. Otherwise go to step 2.


6. If $\text{improvement} = false$, $\text{stopExpand} = false$, and $j \leq m$, then set $\text{stopExpand} := true$ if any $t_p \in \{t_i, t_{i+1}\}, \ldots, t_{j-1}$ must precede $t_j$. Otherwise set $i := i + 1$ and go to step 4.

7. If $\text{stopExpand} = false$, then set $\bar{T} := (\bar{t}_1, \ldots, \bar{t}_m) = \{t_1, \ldots, t_{i-1}, t_j, t_{j-1}, \ldots, t_i, t_{j+1}, \ldots, m\}$ and compute the total cost of $\bar{T}$, $C_{\bar{T}}$. Otherwise go to step 6.

8. If $C_T < C_{\bar{T}}$, then set $T := \bar{T}$, $C_T := C_{\bar{T}}$, $\text{improvement} := true$, and go to step 2. Otherwise set $j := j + 1$ and go to step 6.
6.1.4 Path Inverting 3-opt

We also consider path inverting 3-exchanges. The PI 3-opt algorithm attempts to find two consecutive subsequences within a feasible tour such that the precedence constraints are satisfied both for the case when the two subsequences are swapped and for the case when one of the subsequences is inverted. This is done in a similarly as for the 2-opt counterpart, but the PI 3-opt loops over three indices, h, i, j, and checks twice whether the precedence constraints are fulfilled. As a result the worst case computing time of the PI 3-opt is $O(m^4)$.

6.1.5 Group Swap

Let $T = \{t_1, \ldots, t_m\}$ be a tour with group order $\{G_1, G_2, \ldots, G_m\}$ such that $t_p \in G_p$, $p = 1, \ldots, m$. The Group Swap (GS) algorithm then attempts to swap two groups in the sequence such that the tour remains feasible and becomes improved. Note that the GS exchange is a special case of the double bridge move where $\text{path}_1$ and $\text{path}_2$ consist of only one node each. However, the search for two groups that, when swapped, result in a feasible tour, is equivalent to finding an invertible subsequence $\{t_{p_1}, \ldots, t_{q_1}\}$ and then swapping the groups $G_{p_1}$ and $G_{q_1}$. This means that we can use the same procedure as in the PI 2-opt and that the worst case computing time for the GS algorithm is $O(m^3)$ as compared to $O(m^4)$ for the DB algorithm.

6.1.6 Local Node Selection

Important to note is that all of the above described local search methods deal with the group sequence subproblem of the PCGTSP only. To incorporate an improvement of the node selection in the $k$-opt algorithms a local node selection improvement procedure is executed every time a feasible $k$-exchange is performed. This sort of local improvement has been implemented within $k$-opt algorithms for the GTSP with good results (see [11–13]).

Let $T = \{t_1, \ldots, t_m\}$ be a feasible tour and let $\{G_1, \ldots, G_m\}$ be its group order such that $t_p \in G_p$, $p = 1, \ldots, m$. Finding the optimal node selection for $T$ is then equivalent to finding the shortest path in a layered network constructed by all the possible arcs between the nodes in each consecutive pair of groups in the ordered sequence $\{G_1, \ldots, G_m, G_1\}$ (see Figure 6.6 for an illustration).

6.2 Group Optimization

Given a group sequence, one can optimize the node selection in $O(mp_{\min}p_{\max})$ time when using a dynamic programming approach (see [12]), where $p_{\min} := \min \{|V_p| : p \in G\}$ and $p_{\max} := \max \{|V_p| : p \in G\}$ are the number of nodes in the smallest and largest group respectively.

Let $T = \{t_1, \ldots, t_m\}$ be a tour with group order $\{G_1, G_2, \ldots, G_m\}$ such that $t_p \in G_p$, $p = 1, \ldots, m$. Finding the optimal node selection for $T$ is then equivalent to finding the shortest path in a layered network constructed by all the possible arcs between the nodes in each consecutive pair of groups in the ordered sequence $\{G_1, \ldots, G_m, G_1\}$ (see Figure 6.6 for an illustration).
6.3 Multi-Start Algorithm

A Multi-Start (MS) algorithm is a metaheuristic and an often used framework for local search heuristics. Using a construction heuristic the MS algorithm generates a random initial solution and then applies one or several local methods to the initial solution until a locally optimal

The Group Optimization (GO) algorithm uses a dynamic programming procedure to find $|G_1|$ shortest paths in such a network, and then chooses the shortest of them, thus optimizing the node selection of the given order of groups $\{G_1, \ldots, G_m, G_1\}$. For each $i \in V$, let the variables $C_i$ and $B_i$ represent state variables that keep track of the cost of the shortest path to node $i$, and which node preceded node $i$, i.e. which node was chosen in group $G_i - 1$, in that path respectively. The GO algorithm then works as follows:

**Algorithm 6.3 Group Optimization**

For each $t \in G_1$:

0. Let $G_{m+1} := G_1$.

1. Initialize by setting $C_i = c_{ti}$ and $B_i = t$ for all nodes $i \in G_2$. Set $p = 3$.

2. Compute $C_i = C_h + c_{hi}$ for all nodes $i \in G_p$, where $h$ is the node in $G_{p-1}$ which minimizes $C_i$, and set $B_i = h$.

3. Set $p = p + 1$. If $p \leq m + 1$ go to step 2.

4. Construct the shortest path from node $t \in G_1$ to the corresponding node $\tilde{t} \in G_{m+1}$, denoted $P^t = \{P^t_1, \ldots, P^t_{m+1}\}$, recursively by letting $P^t_j = B_{j+1}$, $j = 1, \ldots, m$, be the $j$:th node in the path and letting $P^t_{m+1} = t$.

Choose the shortest path $P = \{P_1, \ldots, P_{m+1}\}$ as the shortest one among all $P^t$.

The procedure in Algorithm 6.3 has a worst case computing time of $O(mp_{\text{max}}^3)$ but by permuting the sequence of groups such that $|G_1| = p_{\text{min}}$, a computing time of $O(mp_{\text{min}}p_{\text{max}})$ is achieved.

---

Figure 6.6: A layered network representation of a PCGTSP tour ($m = 6$).
solution is found. The procedure is then repeated for a fixed number of initial solutions. The best solution among the locally optimal solutions found is then chosen. It is important that the construction heuristic produces random solutions so that the MS algorithm is able to investigate diverse parts of the set of feasible solutions.

### 6.3.1 Construction Heuristic

Let \( T = \{t_1, \ldots, t_m\} \) be a feasible tour for the PCGTSP with a predetermined start group \( G_{\text{start}} \). The construction heuristic used by the MS algorithm for the PCGTSP works according to the following procedure:

**Algorithm 6.4 Construction heuristic for the PCGTSP**

1. Set \( t_1 \in G_{\text{start}} \) and set \( i = 2 \).
2. Choose a random node \( r \in V \).
3. If \( r \) belongs to a group which has already been visited go to step 2.
4. If \( r \) belongs to a group that is required to be preceded by some other group \( G_p \) which has not yet been visited, set \( r = h \) for some node \( h \in G_p \) and repeat step 4.
5. Set \( t_i := r \). If \( i < m \) then set \( i := i + 1 \) and go to step 2.
6. Apply Algorithm 6.3 to the resulting tour \( T \).

Note that this construction heuristic is based on the assumption that the start group is fixed. For the applications that are considered in this thesis the traveller is assumed to start at a predetermined group \( G_i \) and then return to \( G_i \) when all of the other groups have been visited and thus the start group can always be assumed to be fixed.

### 6.4 Ant Colony System

Ant Colony Optimization (ACO) is a family of metaheuristics which mimics the behaviour of ants to solve optimization problems which can be expressed as shortest path problems. Gambardella and Dorigo [8] and Gambardella et al. [15] have had success when implementing a specific ACO algorithm called Ant Colony System (ACS), which was hybridized with the LPP 3-opt local search, for the SOP. Since the ACO algorithm is a general method for solving optimization problems it can easily be adapted to the PCGTSP.

The idea for the ACS algorithm is to model a fixed number of ants, \( N \), that iteratively generate feasible solutions to the PCGTSP by traversing arcs, \( (i, j) \in A \), in a non-deterministic manner. In each iteration the generation of paths is guided by the depositing of "pheromones", which are denoted \( \tau_{ij} \in [0, 1] \), along the arcs that have been traversed by the ant which has produced the shortest tour. The higher the value of \( \tau_{ij} \), the higher the probability that arc \( (i, j) \) is chosen during the process of generating paths. For each arc \( (i, j) \in A \) a fixed parameter \( \eta_{ij} \in [0, 1] \) is initialized as \( \eta_{ij} = 1/c_{ij} \). This parameter is called the visibility parameter and provides a fixed measurement of how attractive the corresponding arc is for the ants.

The pheromone levels contribute to the intensification of the search. However, to avoid getting stuck at locally optimal solutions and to promote diversification the ACS algorithm incorporates a so-called evaporation rate parameter \( \rho \in [0, 1] \). Let \( \tilde{T}^k = (\tilde{T}^k_1, \ldots, \tilde{T}^k_m) \) be the shortest tour in iteration \( k \). At the end of each iteration \( k \) the pheromone levels are updated
as \( \tau_{ij} = (1 - \rho)\tau_{ij} + \rho/C(\hat{T}^k) \) where \( C(\hat{T}^k) = c(\hat{T}^k_m, \hat{T}^k_1) + \sum_{i=1}^{m} c(\hat{T}^k_i, \hat{T}^k_{i+1}) \) is a cost function where \( c(\hat{T}^k_i, \hat{T}^k_j) \) is the cost associated with the arc \((\hat{T}^k_i, \hat{T}^k_j)\). Furthermore, during the path generation process, if an ant chooses to traverse an arc \((i, j)\), the pheromone level of that arc is updated as \( \tau_{ij} = (1 - \rho)\tau_{ij} + \rho\tau_0 \) where \( \tau_0 \) is the initial pheromone level parameter. The ACS algorithm also introduces a probability \( d_0 \in [0, 1] \) that the arc chosen by an ant during the path generation is the arc which is the most attractive and chooses arcs in proportion to \( \eta_{ij} \) and \( \tau_{ij} \) with probability \((1 - d_0)\).

Let \( \alpha \) and \( \beta \) be parameters that control the relative importance of the pheromone level and the visibility parameter, respectively, when choosing an arc. When adapted to the PCGTSP the procedure for generating paths then works as follows:

**Algorithm 6.5** Path generation for the ACS algorithm

For each ant \( a = 1, \ldots, N \):

1. Initialize the tour by setting the first node, \( T^a_1 \), in the tour of ant \( a \) to a predetermined start node. Set \( h := 2 \).

2. Compute the set of allowed nodes, denoted \( V(T^a) \), by taking into account the precedence constraints and the groups in \( T^a \).

3. Let \( i := T^a_{h-1} \) and let \( d \in [0, 1] \) be random. If \( d > d_0 \) choose to traverse the arc \((i, j)\) with probability

\[
 p^a_{i,j} = \begin{cases} 
 \frac{[\tau_{ij}]^\alpha[\eta_{ij}]^\beta}{\sum_{l \in V(T^a)} [\tau_{il}]^\alpha[\eta_{il}]^\beta} & \text{if } j \in V(T^a) \\
 0 & \text{otherwise}
\end{cases} \tag{6.1}
\]

If \( d \leq d_0 \) then let the next arc be \((i, \tilde{j})\) where \( \tilde{j} := \arg\max_{j \in V(T^a)} \{[\tau_{ij}]^\alpha[\eta_{ij}]^\beta\} \).

4. If arc \((i, j)\) is traversed, then set \( T^a_h := j \) and update its pheromone level according to \( \tau_{ij} := (1 - \rho)\tau_{ij} + \rho\tau_0 \).

5. If \( h \leq m \) set \( h := h + 1 \) and go to step 2.
In this chapter we describe how the different algorithms have been tested and how the test data has been generated.

7.1 Problem Instances

As the PCGTSP has not been extensively studied there does not exist any benchmark problem instances for the PCGTSP. However, there exists several widely used benchmark instances for the SOP with known optimal values for many of them [25]. To create instances the PCGTSP, an SOP instance can be extended by creating a random number of duplicate nodes of every node, letting all the duplicate nodes be a part of the same group and then perturbing all of the arc costs by some percentage. This will create groups with precedence constraints forced upon them and with member nodes whose arc costs differ from one another.

For the purposes of properly evaluating the algorithms described in Chapter 6, fourteen SOP problem instances of various sizes from TSPLIB [25] have been extended to PCGTSP instances. For each one of the SOP instances each node is duplicated \(r\) times, where \(r\) is a uniformly distributed random number on the interval \([10, 20]\). So, if a SOP problem instance has \(m\) nodes, the corresponding PCGTSP problem instance will have \(m\) groups and a minimum of \(10m\) nodes and a maximum of \(20m\) nodes. Each one of the fourteen PCGTSP instances is created in three different versions having different maximum perturbation levels: 20%, 50% and 80%. This means that for the \(p\%\) version of a problem instance all the costs are updated according to 
\[
c_{ij} := c_{ij} \cdot \omega,
\]
where \(\omega\) is a uniformly distributed random number on the interval \([1 - p/100, 1 + p/100]\). The purpose of having different levels of perturbation is to investigate the effect of the level of "clustering" within the groups (i.e. how close the nodes within each group are to each other in relation to the distances between nodes in different groups) on the quality of the solution produced by the algorithms. So a 20\% version of an instance is considered more clustered than the corresponding 50\% version.

While the SOP instances may be based on different real world applications, the data that is extended from SOP instances can no longer be considered as such and will thus be referred to as synthetic. Other than for these synthetic problem instances the metaheuristic algorithms will be tested on four different problem instances generated from a simulation of a CMM measuring points in a couple of different objects. In the problem instances cmm001–cmm003 the CMM is measuring points in three different test objects with different features and in cmm004 the CMM is measuring points in a car body. The instances cmm001–cmm003 are quite small and only have 13–18 groups, and 15–36 nodes while cmm004 has 91 groups and 216 nodes.
7.2 Testing of Heuristic Algorithms

All of our heuristic algorithms have been implemented in C++ and have gone through a total of ten trial runs on a PC with an Intel Core i5-2500 3.30 GHz CPU and 8 GB of RAM.

The local search heuristics will be evaluated within an MS algorithm. First the construction heuristic will construct an initial solution, then the local search heuristic will be applied until a locally optimal solution is found, and finally the GO algorithm will be applied to the locally optimal solution. One trial run will consist of a maximum of 2000 initial solutions and will be terminated if the maximum time of 600 seconds (CPU time) is exceeded or if more than 200 consecutive locally optimal solutions are found without improving the best solution found.

The MS algorithm will go through the same testing procedure as the local search heuristics but will instead use several local search heuristics in succession for each initial solution. The local search heuristics to be used by the MS algorithm will be determined based on the results of the tests done on the local search heuristics.

The ACS algorithm will be hybridized with the same local search heuristics that are implemented for the MS algorithm. To make the ACS algorithm comparable with the MS algorithm it will be tested with ten ants and a maximum of 200 iterations. This will result in a maximum of 2000 generated solutions for each trial run, which is the same as for the MS algorithm. As with the MS algorithm, a trial run for the ACS algorithm will also terminate if the running time exceeds 600 seconds or if more than 20 consecutive iterations are executed without improving the best solution found. The parameters are initialized as $\rho := 0.1$, $\alpha := 1$, $\beta := 2$, $d_0 := 0.9$ and $\tau_0 := 1/(m \cdot z(x_{\text{init}}))$ where $x_{\text{init}}$ is the best locally optimal tour found by the MS algorithm when allowed to search for a maximum of 20 locally optimal solutions.

7.3 Testing of the MILP Model

The proposed MILP model for the PCGTSP is tested on all of the synthetic and CMM problem instances using the generic MILP solving software CPLEX [26]. Each problem instance is provided with an upper bound produced by running the MS algorithm for ten initial solutions. Each problem instance and its LP relaxation is attempted to be solved within a time limit of 24 hours (CPU time).

The LP relaxation of the MILP model will provide a lower bound on the optimal value of the problem instance at hand. However, a tighter bound on the optimal value for a PCGTSP problem instance, which has been created through the procedure described in Section 7.1, may be acquired from the optimal value of the corresponding SOP problem instance. Let $z_{\text{ILP}}$ and $z_{\text{LP}}$ be the optimal value of a PCGTSP instance and its corresponding LP relaxation, and let $z_{\text{SOP}}$ be the optimal value of the corresponding SOP instance. For the $p\%$ version of a PCGTSP instance an alternative lower bound on $z_{\text{ILP}}$ is $M_L := (1 - p/100)z_{\text{SOP}}$. Since the arc costs are decreased by a maximum of $p\%$, the optimal value is also decreased by a maximum of $p\%$. We hence define the lower bound for every synthetic problem instance as $M_L := \max\{M_L, z_{\text{LP}}\}$.
Results

In this chapter the results of our experiments are presented.

For a problem instance let $z^*_{\text{MILP}}$ and $z^*_{\text{LP}}$ denote the optimal values for the instance and its LP relaxation, respectively obtained by running the MILP model through the generic solver CPLEX). Further, let $z^*_{\text{avg}}$ denote the average of the best objective function values found over the ten trial runs for a heuristic algorithm and let $z^*_{\text{best}}$ denote the best objective value found over the ten trial runs. For a problem instance, let $z^*_{\text{LS}}$ denote the best objective function value found by any local search heuristic and let $z^*_{\text{H}}$ denote the best objective function value found by any heuristic algorithm. Moreover, $T_{\text{avg}}$ denotes the average running time of a trial run (averaged over ten trial runs), $T_{\text{MILP}}$ denotes the time required to obtain a optimal solution from CPLEX, and $T_{\text{LP}}$ denotes the time required to solve the corresponding LP relaxation. All computing times are measured in CPU seconds.

8.1 MILP Model

Tables 8.1–8.3 show the results for the synthetic problem instances while Table 8.4 shows the results for the CMM problem instances. The relative gap between $z^*_{\text{MILP}}$ and $z^*_{\text{LP}}$ is called the relative integrality gap and is computed as $(z^*_{\text{MILP}} - z^*_{\text{LP}}) / z^*_{\text{MILP}}$.

<table>
<thead>
<tr>
<th>Instance</th>
<th>$m$</th>
<th>$n$</th>
<th>$z^*_{\text{MILP}}$</th>
<th>$z^*_{\text{LP}}$</th>
<th>$T_{\text{MILP}}$ (s)</th>
<th>$T_{\text{LP}}$ (s)</th>
<th>Gap (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.ESC12</td>
<td>14</td>
<td>194</td>
<td>1360.77</td>
<td>1267.27</td>
<td>8.94</td>
<td>0.13</td>
<td>6.87</td>
</tr>
<tr>
<td>20.ESC25</td>
<td>27</td>
<td>392</td>
<td>1362.55</td>
<td>1327.89</td>
<td>22.75</td>
<td>3.57</td>
<td>2.54</td>
</tr>
<tr>
<td>20.ESC47</td>
<td>49</td>
<td>755</td>
<td>1041.56</td>
<td>1004.62</td>
<td>11327.50</td>
<td>124.09</td>
<td>3.54</td>
</tr>
<tr>
<td>20.ft53.1</td>
<td>54</td>
<td>830</td>
<td>-</td>
<td>4937.32</td>
<td>&gt;86400</td>
<td>189.37</td>
<td>-</td>
</tr>
<tr>
<td>20.ft53.2</td>
<td>54</td>
<td>753</td>
<td>-</td>
<td>5120.79</td>
<td>&gt;86400</td>
<td>358.42</td>
<td>-</td>
</tr>
<tr>
<td>20.ft53.3</td>
<td>54</td>
<td>844</td>
<td>-</td>
<td>6600.65</td>
<td>&gt;86400</td>
<td>397.61</td>
<td>-</td>
</tr>
<tr>
<td>20.ft53.4</td>
<td>54</td>
<td>811</td>
<td>-</td>
<td>10579.20</td>
<td>&gt;86400</td>
<td>39.21</td>
<td>-</td>
</tr>
<tr>
<td>20.ft70.1</td>
<td>71</td>
<td>1091</td>
<td>-</td>
<td>31303.60</td>
<td>&gt;86400</td>
<td>8016.79</td>
<td>-</td>
</tr>
<tr>
<td>20.ft70.2</td>
<td>71</td>
<td>1083</td>
<td>-</td>
<td>31587.00</td>
<td>&gt;86400</td>
<td>5827.20</td>
<td>-</td>
</tr>
<tr>
<td>20.ft70.3</td>
<td>71</td>
<td>1058</td>
<td>-</td>
<td>32871.90</td>
<td>&gt;86400</td>
<td>5660.09</td>
<td>-</td>
</tr>
<tr>
<td>20.ft70.4</td>
<td>71</td>
<td>1060</td>
<td>-</td>
<td>41291.40</td>
<td>&gt;86400</td>
<td>300.53</td>
<td>-</td>
</tr>
<tr>
<td>20.rbg109a</td>
<td>111</td>
<td>1658</td>
<td>-</td>
<td>804.64</td>
<td>&gt;86400</td>
<td>23.66</td>
<td>-</td>
</tr>
<tr>
<td>20.rbg150a</td>
<td>152</td>
<td>2304</td>
<td>-</td>
<td>1386.00</td>
<td>&gt;86400</td>
<td>126.43</td>
<td>-</td>
</tr>
<tr>
<td>20.rbg174a</td>
<td>176</td>
<td>2602</td>
<td>-</td>
<td>1608.26</td>
<td>&gt;86400</td>
<td>261.54</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 8.1: MILP model tested on clustered problem instances (20% version).
8.1. MILP MODEL

<table>
<thead>
<tr>
<th>Instance</th>
<th>$m$</th>
<th>$n$</th>
<th>$z^*_\text{MILP}$</th>
<th>$z^*_\text{LP}$</th>
<th>$T_{\text{MILP}}$ (s)</th>
<th>$T_{\text{LP}}$ (s)</th>
<th>Gap (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50.ESC12</td>
<td>14</td>
<td>206</td>
<td>859.18</td>
<td>805.83</td>
<td>10.17</td>
<td>0.15</td>
<td>6.20</td>
</tr>
<tr>
<td>50.ESC25</td>
<td>27</td>
<td>400</td>
<td>895.38</td>
<td>859.20</td>
<td>27.57</td>
<td>5.55</td>
<td>4.04</td>
</tr>
<tr>
<td>50.ESC47</td>
<td>49</td>
<td>744</td>
<td>673.18</td>
<td>646.68</td>
<td>19100.9</td>
<td>152.67</td>
<td>3.94</td>
</tr>
<tr>
<td>50.ft53.1</td>
<td>54</td>
<td>812</td>
<td>-</td>
<td>3183.83</td>
<td>&gt;86400</td>
<td>333.19</td>
<td>-</td>
</tr>
<tr>
<td>50.ft53.2</td>
<td>54</td>
<td>808</td>
<td>-</td>
<td>3294.37</td>
<td>&gt;86400</td>
<td>433.95</td>
<td>-</td>
</tr>
<tr>
<td>50.ft70.1</td>
<td>71</td>
<td>1067</td>
<td>-</td>
<td>20165.70</td>
<td>&gt;86400</td>
<td>14625.80</td>
<td>-</td>
</tr>
<tr>
<td>50.ft70.2</td>
<td>71</td>
<td>1080</td>
<td>-</td>
<td>20289.40</td>
<td>&gt;86400</td>
<td>11988.30</td>
<td>-</td>
</tr>
<tr>
<td>50.ft70.3</td>
<td>71</td>
<td>1071</td>
<td>-</td>
<td>21128.30</td>
<td>&gt;86400</td>
<td>14451.60</td>
<td>-</td>
</tr>
<tr>
<td>50.ft70.4</td>
<td>71</td>
<td>999</td>
<td>-</td>
<td>26547.20</td>
<td>&gt;86400</td>
<td>709.54</td>
<td>-</td>
</tr>
<tr>
<td>80.ft53.1</td>
<td>54</td>
<td>780</td>
<td>-</td>
<td>1418.94</td>
<td>&gt;86400</td>
<td>619.28</td>
<td>-</td>
</tr>
<tr>
<td>80.ft53.2</td>
<td>54</td>
<td>789</td>
<td>-</td>
<td>1446.07</td>
<td>&gt;86400</td>
<td>813.91</td>
<td>-</td>
</tr>
<tr>
<td>80.ft53.3</td>
<td>54</td>
<td>734</td>
<td>-</td>
<td>1847.80</td>
<td>&gt;86400</td>
<td>571.42</td>
<td>-</td>
</tr>
<tr>
<td>80.ft53.4</td>
<td>54</td>
<td>851</td>
<td>-</td>
<td>2931.41</td>
<td>&gt;86400</td>
<td>77.67</td>
<td>-</td>
</tr>
<tr>
<td>80.ft70.1</td>
<td>71</td>
<td>1017</td>
<td>-</td>
<td>8738.37</td>
<td>&gt;86400</td>
<td>8767.10</td>
<td>-</td>
</tr>
<tr>
<td>80.ft70.2</td>
<td>71</td>
<td>1026</td>
<td>-</td>
<td>8757.97</td>
<td>&gt;86400</td>
<td>15404.00</td>
<td>-</td>
</tr>
<tr>
<td>80.ft70.3</td>
<td>71</td>
<td>1060</td>
<td>-</td>
<td>9038.62</td>
<td>&gt;86400</td>
<td>8959.42</td>
<td>-</td>
</tr>
<tr>
<td>80.ft70.4</td>
<td>71</td>
<td>1051</td>
<td>-</td>
<td>11370.80</td>
<td>&gt;86400</td>
<td>563.04</td>
<td>-</td>
</tr>
<tr>
<td>80.rbg109a</td>
<td>111</td>
<td>1644</td>
<td>-</td>
<td>510.98</td>
<td>&gt;86400</td>
<td>39.79</td>
<td>-</td>
</tr>
<tr>
<td>80.rbg150a</td>
<td>152</td>
<td>2267</td>
<td>-</td>
<td>878.26</td>
<td>&gt;86400</td>
<td>229.98</td>
<td>-</td>
</tr>
<tr>
<td>80.rbg174a</td>
<td>176</td>
<td>2592</td>
<td>-</td>
<td>1017.26</td>
<td>&gt;86400</td>
<td>527.15</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 8.2: MILP model tested on semi-clustered problem instances (50% version).

<table>
<thead>
<tr>
<th>Instance</th>
<th>$m$</th>
<th>$n$</th>
<th>$z^*_\text{MILP}$</th>
<th>$z^*_\text{LP}$</th>
<th>$T_{\text{MILP}}$ (s)</th>
<th>$T_{\text{LP}}$ (s)</th>
<th>Gap (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>80.ESC12</td>
<td>14</td>
<td>248</td>
<td>384.54</td>
<td>347.15</td>
<td>8.66</td>
<td>0.22</td>
<td>9.72</td>
</tr>
<tr>
<td>80.ESC25</td>
<td>27</td>
<td>380</td>
<td>414.31</td>
<td>380.89</td>
<td>109.63</td>
<td>4.84</td>
<td>8.06</td>
</tr>
<tr>
<td>80.ESC47</td>
<td>49</td>
<td>748</td>
<td>303.09</td>
<td>285.73</td>
<td>40389.40</td>
<td>276.30</td>
<td>5.72</td>
</tr>
<tr>
<td>80.ft53.1</td>
<td>54</td>
<td>780</td>
<td>-</td>
<td>1418.94</td>
<td>&gt;86400</td>
<td>619.28</td>
<td>-</td>
</tr>
<tr>
<td>80.ft53.2</td>
<td>54</td>
<td>789</td>
<td>-</td>
<td>1446.07</td>
<td>&gt;86400</td>
<td>813.91</td>
<td>-</td>
</tr>
<tr>
<td>80.ft53.3</td>
<td>54</td>
<td>734</td>
<td>-</td>
<td>1847.80</td>
<td>&gt;86400</td>
<td>571.42</td>
<td>-</td>
</tr>
<tr>
<td>80.ft53.4</td>
<td>54</td>
<td>851</td>
<td>-</td>
<td>2931.41</td>
<td>&gt;86400</td>
<td>77.67</td>
<td>-</td>
</tr>
<tr>
<td>80.ft70.1</td>
<td>71</td>
<td>1017</td>
<td>-</td>
<td>8738.37</td>
<td>&gt;86400</td>
<td>8767.10</td>
<td>-</td>
</tr>
<tr>
<td>80.ft70.2</td>
<td>71</td>
<td>1026</td>
<td>-</td>
<td>8757.97</td>
<td>&gt;86400</td>
<td>15404.00</td>
<td>-</td>
</tr>
<tr>
<td>80.ft70.3</td>
<td>71</td>
<td>1060</td>
<td>-</td>
<td>9038.62</td>
<td>&gt;86400</td>
<td>8959.42</td>
<td>-</td>
</tr>
<tr>
<td>80.ft70.4</td>
<td>71</td>
<td>1051</td>
<td>-</td>
<td>11370.80</td>
<td>&gt;86400</td>
<td>563.04</td>
<td>-</td>
</tr>
<tr>
<td>80.rbg109a</td>
<td>111</td>
<td>1696</td>
<td>-</td>
<td>212.76</td>
<td>&gt;86400</td>
<td>39.41</td>
<td>-</td>
</tr>
<tr>
<td>80.rbg150a</td>
<td>152</td>
<td>2276</td>
<td>-</td>
<td>364.74</td>
<td>&gt;86400</td>
<td>179.10</td>
<td>-</td>
</tr>
<tr>
<td>80.rbg174a</td>
<td>176</td>
<td>2592</td>
<td>-</td>
<td>420.46</td>
<td>&gt;86400</td>
<td>263.62</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 8.3: MILP model tested on non-clustered problem instances (80% version).

<table>
<thead>
<tr>
<th>Instance</th>
<th>$m$</th>
<th>$n$</th>
<th>$z^*_\text{MILP}$</th>
<th>$z^*_\text{LP}$</th>
<th>$T_{\text{MILP}}$ (s)</th>
<th>$T_{\text{LP}}$ (s)</th>
<th>Gap (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmm001</td>
<td>13</td>
<td>15</td>
<td>49.12</td>
<td>48.85</td>
<td>0.02</td>
<td>0.02</td>
<td>0.54</td>
</tr>
<tr>
<td>cmm002</td>
<td>16</td>
<td>25</td>
<td>20.26</td>
<td>7.60</td>
<td>2.24</td>
<td>0.03</td>
<td>62.49</td>
</tr>
<tr>
<td>cmm003</td>
<td>18</td>
<td>36</td>
<td>20.04</td>
<td>11.43</td>
<td>0.43</td>
<td>0.03</td>
<td>42.96</td>
</tr>
<tr>
<td>cmm004</td>
<td>91</td>
<td>216</td>
<td>-</td>
<td>23.00</td>
<td>&gt;86400</td>
<td>5640.67</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 8.4: MILP model tested on the CMM problem instances.

The CPLEX solver was only able to solve the three smallest synthetic problem instances to optimality within the 24 hour limit. A lower level of clustering seems to result in a larger relative integrality gap and longer computing times when solving the ILPs and their corresponding LP relaxations. However, only three problem instances of each version was solved and therefore one can not draw any final conclusions about the relative integrality gap. For the cmm002 and cmm003 problem instances the LP relaxation produces very weak lower bounds, which result in very large relative integrality gaps. For the cmm001 however the LP relaxation produced very
8.2 Local Search Heuristics

Table 8.5 shows the average relative optimality gap of the local search heuristics when tested on all of the synthetic problem instances. Each column shows the relative optimality gap averaged over all ten trial runs and all of the synthetic problem instances of the same version. Table 8.6 shows the average running time of the local search heuristics when tested on the synthetic problem instances. The relative optimality gap for a problem instance is computed as \( \frac{z_{\text{avg}}^* - z_{\text{MILP}}^*}{z_{\text{MILP}}^*} \) if \( z_{\text{MILP}}^* \) is available, otherwise it is computed as \( \frac{z_{\text{avg}}^* - M_L}{M_L} \). For more detailed results from the experiments on the local search heuristics consult Tables A.4–A.6 in the Appendix.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Avg. Gap (clustered, 20%)</th>
<th>Avg. Gap (semi-clustered, 50%)</th>
<th>Avg. Gap (non-clustered, 80%)</th>
<th>Avg. Gap (total)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PI 2-opt</td>
<td>79.93 %</td>
<td>125.30 %</td>
<td>191.93 %</td>
<td>132.39 %</td>
</tr>
<tr>
<td>PI 3-opt</td>
<td>31.02 %</td>
<td>55.24 %</td>
<td>115.57 %</td>
<td>67.28 %</td>
</tr>
<tr>
<td>LPP 3-opt</td>
<td>5.35 %</td>
<td>23.26 %</td>
<td>66.55 %</td>
<td>32.39 %</td>
</tr>
<tr>
<td>Group Swap</td>
<td>54.03 %</td>
<td>87.81 %</td>
<td>160.86 %</td>
<td>100.90 %</td>
</tr>
<tr>
<td>Double Bridge</td>
<td>19.28 %</td>
<td>41.45 %</td>
<td>95.82 %</td>
<td>52.18 %</td>
</tr>
</tbody>
</table>

Table 8.5: Average optimality gaps resulting from the local search heuristics tested on the synthetic problem instances.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Avg. Time (clustered, 20%)</th>
<th>Avg. Time (semi-clustered, 50%)</th>
<th>Avg. Time (non-clustered, 80%)</th>
<th>Avg. Time (total)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PI 2-opt</td>
<td>122.74</td>
<td>90.53</td>
<td>90.21</td>
<td>101.16</td>
</tr>
<tr>
<td>PI 3-opt</td>
<td>416.46</td>
<td>358.48</td>
<td>260.32</td>
<td>345.08</td>
</tr>
<tr>
<td>LPP 3-opt</td>
<td>138.01</td>
<td>127.49</td>
<td>126.35</td>
<td>130.62</td>
</tr>
<tr>
<td>Group Swap</td>
<td>90.28</td>
<td>73.53</td>
<td>74.52</td>
<td>79.44</td>
</tr>
<tr>
<td>Double Bridge</td>
<td>315.26</td>
<td>252.30</td>
<td>241.77</td>
<td>279.78</td>
</tr>
</tbody>
</table>

Table 8.6: Average computing times for local search heuristics tested on the synthetic problem instances.

From Table 8.5 it is apparent that the average optimality gap tends to increase as the level of clustering decreases for all local search heuristics. It also clearly shows that the LPP 3-opt algorithm consistently outperforms the other local search heuristics with respect to solution quality. Table 8.6 demonstrates the efficiency of the LPP 3-opt algorithm as it is very fast while retaining relatively good solution quality. Only the PI 2-opt and GS algorithms are faster but their solution quality is significantly worse. The PI 3-opt is not the worst with respect to solution quality but it is the slowest. While the GS algorithm is the fastest it produces very bad upper bounds.

To more accurately compare the performance of the local search heuristics for the individual problem instances we plot \( z_{\text{avg}}^*/z_{\text{LS}}^* \) for each algorithm and problem instance in Figures 8.1–8.6. A data point being close to constant 1 means that the average solution quality produced by that algorithm is close to the best produced by any of the local search heuristics implemented, for that specific problem instance.
8.2. LOCAL SEARCH HEURISTICS

Figure 8.1: Local search heuristics average solution value divided by the value of the best solution found by any local search heuristic. For the clustered synthetic problem instances.

Figure 8.2: Local search heuristics average solution value divided by the value of the best solution found by any local search heuristic. For the clustered synthetic problem instances (ESCxx excluded). A close-up of the graphs reported in Figure 8.1.
8.2. LOCAL SEARCH HEURISTICS

Figure 8.3: Local search heuristics average solution value divided by the value of the best solution found by any local search heuristic. For the semi-clustered synthetic problem instances.

Figure 8.4: Local search heuristics average solution value divided by the value of the best solution found by any local search heuristic. For the semi-clustered synthetic problem instances (ESCxx excluded). A close-up of the graphs reported in Figure 8.3.
8.2. LOCAL SEARCH HEURISTICS

Figure 8.5: Local search heuristics average solution value divided by the value of the best solution found by any local search heuristic. For the non-clustered synthetic problem instances.

Figure 8.6: Local search heuristics average solution value divided by the value of the best solution found by any local search heuristic. For the non-clustered synthetic problem instances (ESCxx excluded). A close-up of the graphs reported in Figure 8.5.
Figures 8.1, 8.3 and 8.5 illustrate how the deviation from $z^*_{LS}$ becomes inflated as the level of clustering decreases. It also seems like the ESC25 and ESC47 problem instances are particularly hard to solve for most of the local search heuristics. In Figures 8.2, 8.4 and 8.6 the three first problem instances (i.e., ESC12, ESC25, and ESC47) are excluded and the performance of the algorithms for the rest of the problem instances is more clearly visible. However, the general trend seems to confirm the results in Table 8.5, i.e. the LPP 3-opt is the most efficient algorithm and the relative optimality gap seems to increase as the level of clustering decreases.

## 8.3 Metaheuristics

The results from the testing of the local search heuristics show that the PI 2-opt and PI 3-opt are clearly worse at producing good quality solutions when compared to the other algorithms. Therefore the path inverting methods are excluded during the testing of the metaheuristics. The local search heuristics are implemented in the MS algorithm after the construction heuristic and in the ACS algorithm after the path generation and are executed in the following order: LPP 3-opt, Group Swap, and Double Bridge. After the local search heuristics have found a locally optimal solution the Group Optimization algorithm is applied to this solution. The GS algorithm is included despite its bad performance since it is fast and can perform some simpler double bridge moves before the DB algorithm is executed and thus potentially decreasing the computing time of the DB algorithm.

When testing on the synthetic problem instances the metaheuristics will be compared with the results of the LPP 3-opt local search heuristic to determine whether the use of metaheuristics show any significant improvement compared to using only the LPP 3-opt with the MS algorithm. The MS algorithm—which used with LPP 3-opt, GS and DB—will be abbreviated MS+LS and when only used with LPP 3-opt it will be abbreviated as MS+LPP3-opt.

Table 8.9 shows the performance of the metaheuristics when tested on the CMM problem instances. Table 8.7 shows average optimality gaps and Table 8.8 shows average computing times when tested on synthetic problem instances. For more detailed results from the experiments on the metaheuristics consult Tables A.1–A.3 in the Appendix.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Avg. Gap (clustered, 20%)</th>
<th>Avg. Gap (semi-clustered, 50%)</th>
<th>Avg. Gap (non-clustered, 80%)</th>
<th>Avg. Gap (total)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MS+LPP 3-opt</td>
<td>7.35 %</td>
<td>23.26 %</td>
<td>66.55 %</td>
<td>32.39 %</td>
</tr>
<tr>
<td>MS+LS</td>
<td>7.04 %</td>
<td>22.08 %</td>
<td>64.52 %</td>
<td>31.21 %</td>
</tr>
<tr>
<td>ACS+LS</td>
<td>8.96 %</td>
<td>21.34 %</td>
<td>41.77 %</td>
<td>24.02 %</td>
</tr>
</tbody>
</table>

Table 8.7: Average optimality gap for the metaheuristics and the LPP 3-opt algorithm tested on the synthetic problem instances.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Avg. Time (clustered, 20%)</th>
<th>Avg. Time (semi-clustered, 50%)</th>
<th>Avg. Time (non-clustered, 80%)</th>
<th>Avg. Time (total)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MS+LPP 3-opt</td>
<td>138.01</td>
<td>127.49</td>
<td>126.35</td>
<td>130.62</td>
</tr>
<tr>
<td>MS+LS</td>
<td>240.82</td>
<td>251.83</td>
<td>235.90</td>
<td>242.85</td>
</tr>
<tr>
<td>ACS+LS</td>
<td>412.55</td>
<td>395.74</td>
<td>359.27</td>
<td>389.19</td>
</tr>
</tbody>
</table>

Table 8.8: Average computing times for the metaheuristics and the LPP 3-opt algorithm tested on the synthetic problem instances.
### 8.3. METAHEURISTICS

<table>
<thead>
<tr>
<th>Instance</th>
<th>m</th>
<th>n</th>
<th>$z_{\text{ILP}}^*$</th>
<th>$z_P^*$</th>
<th>$T_{\text{avg}}$ (s)</th>
<th>$z_{\text{avg}}^*$</th>
<th>$z_{\text{best}}^*$</th>
<th>Gap (%)</th>
<th>$T_{\text{avg}}$ (s)</th>
<th>$z_{\text{avg}}^*$</th>
<th>$z_{\text{best}}^*$</th>
<th>Gap (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmm001</td>
<td>13</td>
<td>15</td>
<td>49.12</td>
<td>48.85</td>
<td>2.34</td>
<td>49.12</td>
<td>49.12</td>
<td>0.00</td>
<td>3.35</td>
<td>49.12</td>
<td>49.12</td>
<td>0.00</td>
</tr>
<tr>
<td>cmm002</td>
<td>16</td>
<td>25</td>
<td>20.26</td>
<td>7.60</td>
<td>4.06</td>
<td>20.26</td>
<td>20.26</td>
<td>0.00</td>
<td>5.13</td>
<td>20.68</td>
<td>20.26</td>
<td>2.07</td>
</tr>
<tr>
<td>cmm003</td>
<td>18</td>
<td>36</td>
<td>20.04</td>
<td>11.43</td>
<td>3.33</td>
<td>20.04</td>
<td>20.04</td>
<td>0.00</td>
<td>7.40</td>
<td>20.08</td>
<td>20.04</td>
<td>0.20</td>
</tr>
<tr>
<td>cmm004</td>
<td>91</td>
<td>216</td>
<td>-23.00</td>
<td>-</td>
<td>&gt;600</td>
<td>580.30</td>
<td>51.92</td>
<td>50.27</td>
<td>125.74</td>
<td>&gt;600</td>
<td>47.13</td>
<td>46.41</td>
</tr>
</tbody>
</table>

Table 8.9: Metaheuristics tested on CMM problem instances.

As shown for the local search heuristics in Table 8.5, the trend of an increasing optimality gap as the level of clustering decreases (i.e. as the level of perturbation of the arc costs increases) can also be seen for the metaheuristics in Table 8.7. It also seems like the GS and DB algorithms improve the average solution quality of the MS+LS algorithm only slightly in comparison with the MS+LPP3-opt algorithm when averaged over all problem instances. The average solution quality of the ACS algorithm is slightly worse for the clustered instances, slightly better for the semi-clustered instances, and significantly better for the non-clustered instances.

The computing time for the MS+LS algorithm is almost twice as long as for the MS+LPP3-opt algorithm. For the ACS algorithm, the computing time is about three times as long as for the LPP 3-opt algorithm.

To compare the three methods we plot $z_{\text{avg}}^*/z_{\text{H}}^*$ to compare their average solution quality with the best solution found by any of the heuristic algorithms implemented.

Figure 8.7: Average solution values from the metaheuristics and the LPP 3-opt normalized by the best solution found by any heuristic algorithm; for each of the clustered (20%) synthetic problem instances.
8.3. METAHEURISTICS

Figure 8.8: Average solution values from the metaheuristics and the LPP 3-opt normalized by the best solution found by any heuristic algorithm; for each of the semi-clustered (50%) synthetic problem instances.

Figure 8.9: Average solution values from the metaheuristics and the LPP 3-opt normalized by the best solution found by any heuristic algorithm; for each of the non-clustered (80%) synthetic problem instances.
The results illustrated in Figures 8.7–8.9 show that for the clustered instances (the only exception being ESC25) the ACS+LS algorithm performs consistently worse than the MS algorithms, for many of the semi-clustered instances the ACS+LS algorithm performs better than the MS algorithms, and for the non-clustered instances the ACS+LS algorithm clearly performs better than the MS algorithms. The difference between the performance of the MS+LS and MS+LPP3-opt algorithms is small but increases as the level of clustering decreases.

Table 8.9 shows that the smaller CMM problem instances (cmm001–cmm003) are almost consistently solved to optimality by both the ACS algorithm and the MS algorithm. However, the result from the MILP model (seen in Table 8.4) shows that the CPLEX optimizing software solves the smaller instances cmm001–cmm003 to optimality much faster than the metaheuristics. While CPLEX did not find an optimal solution for the larger instance cmm004 within the time limit, the metaheuristics produced solutions with quite large optimality gaps as well. The ACS+LS algorithm produces better solutions than the MS+LS algorithm for the largest CMM problem instance which is consistent with the idea that ACS algorithm performs better for non-clustered problem instances.
Discussion and Conclusions

The CPLEX solver was able to find optimal values for the three smallest CMM problem instances much faster than any of the heuristic algorithms. However, for two of the CMM instances the proposed MILP model produced rather weak lower bounds which indicates that the lower bounds produced by the LP relaxation of the model are not tight. The CPLEX solver was only able to solve the three smallest synthetic instances of each version which makes a general evaluation of the proposed MILP model hard. However, the results from both the synthetic and the CMM instances indicate that both the level of clustering and the size of a problem instance have effects on the relative integrality gap. However, further analysis of the MILP model is needed to draw any final conclusions.

All of the heuristic algorithms perform worse with respect to solution quality as the level of clustering decreases. The node selection subproblem becomes increasingly more important as the level of clustering decreases and thus the algorithms’ poor performance when attempting to solve problem instances with a low level of clustering can be attributed to them not dealing with the node selection subproblem in a satisfying manner. This is further supported by the fact that the ACS algorithm produces better quality solutions than the MS algorithm for the non-clustered problem instances. Since the ACS algorithm provides another way of improving node selection with its diversification and intensification features, it performs better for the problem instances with low level of clustering, while the MS algorithm only utilizes the local node selection improvement procedure and the GO algorithm to improve the node selection.

The difference between using only the LPP 3-opt local search heuristic and using the LPP 3-opt, the GS, and the DB local search heuristics together with the MS metaheuristic is very small with respect to solution quality, while the difference in computing times between the two algorithms is very large. Therefore, one should either entirely exclude the GS and the DB algorithms from the local search phase of the metaheuristics or attempt to improve their implementations so that their computing times are decreased.

For real applications such as the CMM instances it seems like the smaller problem instances can be solved to optimality within a very short time by both the CPLEX solver and the heuristic algorithms, with the CPLEX solver being much faster. However, for the largest problem instance the MS and ACS algorithms produced solutions with very large optimality gaps. The cause of the poor performance is most likely due to the fact that the relative integrality gap is very high for that problem instance. This is supported by the facts that the relative integrality gap has been shown to be quite large for two other CMM instances and that the metaheuristic algorithms have not been shown to have such poor performance for any other problem instance.
Future Research

Suggestions for future research of the PCGTSP are improvements of the proposed MILP model by strengthening the formulation with valid inequalities and further analysis of its polytope. The results from the experiments suggest that generic solution strategies such as the ones employed by CPLEX are not enough to solve large instances of the PCGTSP within a reasonable time frame and thus more problem specific optimizing methods are needed to solve them.

As for the heuristic algorithms, the experiments reveal a need for improvement of the node selection components within the $k$-opt algorithms as well as some general strategy for the node selection subproblem within the metaheuristics. There are also many other algorithms and strategies that are not considered in this thesis that are regularly employed when solving the TSP and its variations. Examples of such algorithms are Particle Swarm Optimization metaheuristics (see [16]), Tabu Search metaheuristics (see [23]), and more advanced Lin-Kernighan adaptations (see [11, 24]). Some of these may be adapted to the PCGTSP.

One can also consider the multi-agent PCGTSP, in which the multiple travellers would correspond to multiple robot arms working on the same object. In the case of CMM measuring for example, using several measuring probes decreases the overall measuring time and is sometimes required to reach all of the points in the object.
References


REFERENCES


### Appendix

#### Table A.1: Results from the metaheuristics tested on the clustered problem instances (20% version).

<table>
<thead>
<tr>
<th>Instance</th>
<th>$z^*_{MILP}$</th>
<th>$M_L$</th>
<th>$T_{avg}$ (s)</th>
<th>$z_{avg}$</th>
<th>$z^*_{best}$</th>
<th>Gap (%)</th>
<th>$T_{avg}$ (s)</th>
<th>$z_{avg}$</th>
<th>$z^*_{best}$</th>
<th>Gap (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.ESC12</td>
<td>1360.77</td>
<td>1340.00</td>
<td>9.79</td>
<td>1360.77</td>
<td>1360.77</td>
<td>0.00</td>
<td>15.00</td>
<td>1360.77</td>
<td>1360.77</td>
<td>0.00</td>
</tr>
<tr>
<td>20.ESC25</td>
<td>1362.55</td>
<td>1344.80</td>
<td>50.54</td>
<td>1393.14</td>
<td>1362.55</td>
<td>2.25</td>
<td>65.41</td>
<td>1362.00</td>
<td>1362.55</td>
<td>0.25</td>
</tr>
<tr>
<td>20.ESC47</td>
<td>1041.56</td>
<td>1030.40</td>
<td>219.62</td>
<td>1270.04</td>
<td>1240.86</td>
<td>21.94</td>
<td>371.28</td>
<td>1324.97</td>
<td>1219.37</td>
<td>27.21</td>
</tr>
<tr>
<td>20.ft53.1</td>
<td>6024.80</td>
<td>274.93</td>
<td>6380.39</td>
<td>6246.75</td>
<td>5.90</td>
<td>462.70</td>
<td>6409.48</td>
<td>6294.81</td>
<td>6.38</td>
<td></td>
</tr>
<tr>
<td>20.ft53.2</td>
<td>6420.80</td>
<td>149.19</td>
<td>6708.82</td>
<td>6708.82</td>
<td>8.06</td>
<td>294.14</td>
<td>7336.22</td>
<td>6782.79</td>
<td>14.26</td>
<td></td>
</tr>
<tr>
<td>20.ft53.3</td>
<td>8209.60</td>
<td>114.39</td>
<td>8904.88</td>
<td>8715.47</td>
<td>8.47</td>
<td>268.47</td>
<td>9324.50</td>
<td>8936.98</td>
<td>13.58</td>
<td></td>
</tr>
<tr>
<td>20.ft53.4</td>
<td>11540.00</td>
<td>73.70</td>
<td>11977.00</td>
<td>11879.00</td>
<td>3.79</td>
<td>294.14</td>
<td>12009.10</td>
<td>11906.10</td>
<td>27.21</td>
<td></td>
</tr>
<tr>
<td>20.rbg109a</td>
<td>830.40</td>
<td>289.35</td>
<td>907.91</td>
<td>891.31</td>
<td>4.47</td>
<td>600.00</td>
<td>9324.50</td>
<td>8936.98</td>
<td>13.58</td>
<td></td>
</tr>
</tbody>
</table>

#### Table A.2: Results from the metaheuristics tested on the semi-clustered problem instances (50% version).

<table>
<thead>
<tr>
<th>Instance</th>
<th>$z^*_{MILP}$</th>
<th>$M_L$</th>
<th>$T_{avg}$ (s)</th>
<th>$z_{avg}$</th>
<th>$z^*_{best}$</th>
<th>Gap (%)</th>
<th>$T_{avg}$ (s)</th>
<th>$z_{avg}$</th>
<th>$z^*_{best}$</th>
<th>Gap (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50.ESC12</td>
<td>859.18</td>
<td>837.50</td>
<td>14.38</td>
<td>870.90</td>
<td>859.18</td>
<td>1.36</td>
<td>20.00</td>
<td>871.96</td>
<td>859.18</td>
<td>1.49</td>
</tr>
<tr>
<td>50.ESC25</td>
<td>895.38</td>
<td>840.50</td>
<td>39.16</td>
<td>964.88</td>
<td>903.01</td>
<td>7.68</td>
<td>64.50</td>
<td>920.63</td>
<td>895.38</td>
<td>2.82</td>
</tr>
<tr>
<td>50.ESC47</td>
<td>673.18</td>
<td>644.00</td>
<td>273.76</td>
<td>920.72</td>
<td>837.41</td>
<td>36.77</td>
<td>379.50</td>
<td>960.15</td>
<td>891.31</td>
<td>42.63</td>
</tr>
<tr>
<td>50.ft53.1</td>
<td>3765.50</td>
<td>256.95</td>
<td>4604.83</td>
<td>4373.27</td>
<td>22.29</td>
<td>428.82</td>
<td>4946.67</td>
<td>4421.45</td>
<td>24.73</td>
<td></td>
</tr>
<tr>
<td>50.ft53.2</td>
<td>4013.00</td>
<td>177.21</td>
<td>5181.47</td>
<td>4999.81</td>
<td>29.12</td>
<td>390.56</td>
<td>5236.72</td>
<td>5079.63</td>
<td>30.49</td>
<td></td>
</tr>
<tr>
<td>50.ft53.3</td>
<td>5131.00</td>
<td>100.87</td>
<td>6675.95</td>
<td>6511.27</td>
<td>30.11</td>
<td>252.40</td>
<td>6904.14</td>
<td>6437.50</td>
<td>34.56</td>
<td></td>
</tr>
<tr>
<td>50.ft53.4</td>
<td>7212.50</td>
<td>80.26</td>
<td>8601.33</td>
<td>8510.95</td>
<td>19.26</td>
<td>291.01</td>
<td>8316.29</td>
<td>8013.63</td>
<td>15.30</td>
<td></td>
</tr>
<tr>
<td>50.rbg109a</td>
<td>519.00</td>
<td>306.19</td>
<td>697.40</td>
<td>598.84</td>
<td>17.03</td>
<td>600.00</td>
<td>7242.93</td>
<td>7074.50</td>
<td>20.73</td>
<td></td>
</tr>
<tr>
<td>50.rbg150a</td>
<td>875.00</td>
<td>525.66</td>
<td>1477.82</td>
<td>1444.13</td>
<td>4.13</td>
<td>600.00</td>
<td>1461.50</td>
<td>1400.00</td>
<td>5.82</td>
<td></td>
</tr>
<tr>
<td>50.rbg174a</td>
<td>1400.00</td>
<td>489.71</td>
<td>1457.82</td>
<td>1444.13</td>
<td>4.13</td>
<td>&gt;600</td>
<td>1461.50</td>
<td>1400.00</td>
<td>5.82</td>
<td></td>
</tr>
<tr>
<td>Instance</td>
<td>$z^*_\text{MLP}$</td>
<td>$M_L$</td>
<td>$T_{\text{avg}}$ (s)</td>
<td>$z^*_\text{avg}$</td>
<td>$z^*_{\text{best}}$</td>
<td>Gap (%)</td>
<td>$T_{\text{avg}}$ (s)</td>
<td>$z^*_\text{avg}$</td>
<td>$z^*_{\text{best}}$</td>
<td>Gap (%)</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------</td>
<td>------</td>
<td>---------------------</td>
<td>------------------</td>
<td>-------------------</td>
<td>---------</td>
<td>---------------------</td>
<td>------------------</td>
<td>-------------------</td>
<td>---------</td>
</tr>
<tr>
<td>80.ESC12</td>
<td>384.54</td>
<td>335.00</td>
<td>17.30</td>
<td>427.12</td>
<td>411.28</td>
<td>11.07</td>
<td>21.87</td>
<td>404.64</td>
<td>389.2</td>
<td>5.23</td>
</tr>
<tr>
<td>80.ESC25</td>
<td>414.31</td>
<td>336.20</td>
<td>35.28</td>
<td>545.90</td>
<td>501.75</td>
<td>31.76</td>
<td>55.69</td>
<td>440.55</td>
<td>435.03</td>
<td>6.33</td>
</tr>
<tr>
<td>80.ESC47</td>
<td>303.09</td>
<td>257.60</td>
<td>235.24</td>
<td>516.76</td>
<td>469.31</td>
<td>70.50</td>
<td>322.54</td>
<td>514.14</td>
<td>434.71</td>
<td>69.63</td>
</tr>
<tr>
<td>80.ft53.1</td>
<td></td>
<td>1506.20</td>
<td>242.14</td>
<td>2756.13</td>
<td>2560.44</td>
<td>82.99</td>
<td>270.97</td>
<td>2393.79</td>
<td>2317.80</td>
<td>58.93</td>
</tr>
<tr>
<td>80.ft53.2</td>
<td></td>
<td>1605.20</td>
<td>146.13</td>
<td>3074.72</td>
<td>2987.14</td>
<td>31.55</td>
<td>340.70</td>
<td>2549.42</td>
<td>2461.79</td>
<td>58.82</td>
</tr>
<tr>
<td>80.ft53.3</td>
<td></td>
<td>2052.40</td>
<td>78.58</td>
<td>3849.38</td>
<td>3607.18</td>
<td>87.56</td>
<td>210.76</td>
<td>3435.57</td>
<td>3318.79</td>
<td>67.39</td>
</tr>
<tr>
<td>80.ft53.4</td>
<td></td>
<td>2885.00</td>
<td>69.89</td>
<td>4646.95</td>
<td>4432.34</td>
<td>61.07</td>
<td>220.58</td>
<td>3841.54</td>
<td>3712.28</td>
<td>34.65</td>
</tr>
<tr>
<td>80.ft70.1</td>
<td></td>
<td>7862.00</td>
<td>543.41</td>
<td>14046.30</td>
<td>13487.10</td>
<td>48.65</td>
<td>480.66</td>
<td>11232.10</td>
<td>11107.10</td>
<td>42.85</td>
</tr>
<tr>
<td>80.ft70.2</td>
<td></td>
<td>8020.20</td>
<td>330.51</td>
<td>14910.70</td>
<td>14555.30</td>
<td>85.91</td>
<td>396.48</td>
<td>11597.20</td>
<td>11390.30</td>
<td>44.60</td>
</tr>
<tr>
<td>80.ft70.3</td>
<td></td>
<td>8507.00</td>
<td>147.46</td>
<td>15573.10</td>
<td>15034.60</td>
<td>83.06</td>
<td>482.63</td>
<td>12311.10</td>
<td>11915.60</td>
<td>44.72</td>
</tr>
<tr>
<td>80.ft70.4</td>
<td></td>
<td>10706.00</td>
<td>121.39</td>
<td>16467.30</td>
<td>16258.70</td>
<td>53.81</td>
<td>417.92</td>
<td>15114.50</td>
<td>14701.10</td>
<td>41.18</td>
</tr>
<tr>
<td>80.rbg109a</td>
<td></td>
<td>207.60</td>
<td>293.52</td>
<td>346.17</td>
<td>335.40</td>
<td>66.75</td>
<td>&gt;600</td>
<td>294.79</td>
<td>286.81</td>
<td>42.00</td>
</tr>
<tr>
<td>80.rbg150a</td>
<td></td>
<td>350.00</td>
<td>457.74</td>
<td>519.26</td>
<td>496.40</td>
<td>48.36</td>
<td>&gt;600</td>
<td>471.45</td>
<td>456.95</td>
<td>34.70</td>
</tr>
<tr>
<td>80.rbg174a</td>
<td></td>
<td>406.60</td>
<td>584.00</td>
<td>610.65</td>
<td>598.18</td>
<td>50.18</td>
<td>&gt;600</td>
<td>543.55</td>
<td>535.00</td>
<td>33.68</td>
</tr>
</tbody>
</table>

Table A.3: Results from the metaheuristics tested on the non-clustered problem instances (80% version).
| Instance | z_{MLP}^T | \bar{z}_{avg}^T | \bar{z}_{best}^T | \bar{z}_{avg}^T | \bar{z}_{best}^T | \bar{z}_{avg}^T | \bar{z}_{best}^T | \bar{z}_{avg}^T | \bar{z}_{best}^T | \bar{z}_{avg}^T | \bar{z}_{best}^T | \bar{z}_{avg}^T | \bar{z}_{best}^T | \bar{z}_{avg}^T | \bar{z}_{best}^T |
|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| 20.ESC12 | 1360.77  | 1340.00  | 8.96     | 1361.47  | 1360.77  | 8.18     | 1374.26  | 1360.77  | 8.18     | 1374.26  | 1360.77  | 8.18     | 1374.26  | 1360.77  | 8.18     | 1374.26  |
| 20.ESC25 | 1362.55  | 1344.80  | 28.88    | 1361.47  | 1360.77  | 8.18     | 1374.26  | 1360.77  | 8.18     | 1374.26  | 1360.77  | 8.18     | 1374.26  | 1360.77  | 8.18     | 1374.26  |
| 20.ESC47 | 1041.56  | 1030.40  | 89.39    | 1086.50  | 1068.00  | 36.54    | 1180.76  | 1086.50  | 36.54    | 1180.76  | 1086.50  | 36.54    | 1180.76  | 1086.50  | 36.54    | 1180.76  |
| 20.ESC51 | 6024.00  | 6047.60  | 28.88    | 6082.40  | 6007.60  | 36.54    | 6216.00  | 6082.40  | 36.54    | 6216.00  | 6082.40  | 36.54    | 6216.00  | 6082.40  | 36.54    | 6216.00  |
| 20.ESC52 | 6420.40  | 6587.60  | 28.88    | 6542.00  | 6507.60  | 36.54    | 6682.40  | 6542.00  | 36.54    | 6682.40  | 6542.00  | 36.54    | 6682.40  | 6542.00  | 36.54    | 6682.40  |
| 20.ESC53 | 8209.60  | 8296.00  | 28.88    | 8283.20  | 8262.40  | 36.54    | 8368.80  | 8283.20  | 36.54    | 8368.80  | 8283.20  | 36.54    | 8368.80  | 8283.20  | 36.54    | 8368.80  |
| 20.ESC54 | 1140.00  | 1209.60  | 28.88    | 1140.00  | 1209.60  | 36.54    | 1209.60  | 1140.00  | 36.54    | 1209.60  | 1140.00  | 36.54    | 1209.60  | 1140.00  | 36.54    | 1209.60  |
| 20.ESC71 | 3145.00  | 3159.60  | 28.88    | 3145.00  | 3159.60  | 36.54    | 3159.60  | 3145.00  | 36.54    | 3159.60  | 3145.00  | 36.54    | 3159.60  | 3145.00  | 36.54    | 3159.60  |
| 20.ESC72 | 3208.00  | 3264.00  | 28.88    | 3208.00  | 3264.00  | 36.54    | 3264.00  | 3208.00  | 36.54    | 3264.00  | 3208.00  | 36.54    | 3264.00  | 3208.00  | 36.54    | 3264.00  |
| 20.ESC73 | 3402.00  | 3482.00  | 28.88    | 3402.00  | 3482.00  | 36.54    | 3482.00  | 3402.00  | 36.54    | 3482.00  | 3402.00  | 36.54    | 3482.00  | 3402.00  | 36.54    | 3482.00  |
| 20.ESC74 | 4282.00  | 4582.00  | 28.88    | 4282.00  | 4582.00  | 36.54    | 4582.00  | 4282.00  | 36.54    | 4582.00  | 4282.00  | 36.54    | 4582.00  | 4282.00  | 36.54    | 4582.00  |
| 20.rbg109a| 830.40  | 1141.51  | 28.88    | 830.40  | 1141.51  | 36.54    | 1141.51  | 830.40  | 36.54    | 1141.51  | 830.40  | 36.54    | 1141.51  | 830.40  | 36.54    | 1141.51  |
| 20.rbg150a | 1400.00 | 1581.00  | 28.88    | 1400.00 | 1581.00  | 36.54    | 1581.00  | 1400.00 | 36.54    | 1581.00  | 1400.00 | 36.54    | 1581.00  | 1400.00 | 36.54    | 1581.00  |
| 20.rbg174a | 1626.40 | 1771.48  | 28.88    | 1626.40 | 1771.48  | 36.54    | 1771.48  | 1626.40 | 36.54    | 1771.48  | 1626.40 | 36.54    | 1771.48  | 1626.40 | 36.54    | 1771.48  |

Table A.4: The local search heuristics tested on the clustered problem instances (20% version).
<table>
<thead>
<tr>
<th>Instance</th>
<th>$z^*_MILP$</th>
<th>$M_L$</th>
<th>$T_{avg}$</th>
<th>$z^*_avg$</th>
<th>$z^*_best$</th>
<th>LPP 3-opt</th>
<th>$T_{avg}$</th>
<th>$z^*_avg$</th>
<th>$z^*_best$</th>
<th>Group Swap</th>
<th>$T_{avg}$</th>
<th>$z^*_avg$</th>
<th>$z^*_best$</th>
<th>Double Bridge</th>
</tr>
</thead>
<tbody>
<tr>
<td>80.ESC12</td>
<td>384.54</td>
<td>335.00</td>
<td>16.78</td>
<td>500.77</td>
<td>418.65</td>
<td>15.91</td>
<td>424.17</td>
<td>412.88</td>
<td>15.11</td>
<td>416.36</td>
<td>410.01</td>
<td>18.59</td>
<td>459.87</td>
<td>417.69</td>
</tr>
<tr>
<td>80.ESC25</td>
<td>414.31</td>
<td>336.20</td>
<td>22.10</td>
<td>1742.74</td>
<td>1575.16</td>
<td>61.60</td>
<td>922.07</td>
<td>734.28</td>
<td>99.11</td>
<td>516.75</td>
<td>491.75</td>
<td>45.31</td>
<td>2319.12</td>
<td>1909.99</td>
</tr>
<tr>
<td>80.ESC47</td>
<td>257.60</td>
<td>53.65</td>
<td>2829.49</td>
<td>3184.17</td>
<td>2892.49</td>
<td>499.38</td>
<td>1304.69</td>
<td>1082.53</td>
<td>9.23</td>
<td>616.32</td>
<td>593.66</td>
<td>307.78</td>
<td>879.33</td>
<td>817.14</td>
</tr>
<tr>
<td>80.ESC3.1</td>
<td>-</td>
<td>1506.20</td>
<td>50.57</td>
<td>5405.71</td>
<td>5941.12</td>
<td>453.85</td>
<td>4137.21</td>
<td>3905.53</td>
<td>122.67</td>
<td>2807.67</td>
<td>2699.22</td>
<td>46.80</td>
<td>4943.81</td>
<td>4556.53</td>
</tr>
<tr>
<td>80.ESC3.2</td>
<td>-</td>
<td>1605.20</td>
<td>42.73</td>
<td>5382.91</td>
<td>5257.89</td>
<td>203.26</td>
<td>4445.66</td>
<td>4146.43</td>
<td>76.74</td>
<td>3165.87</td>
<td>3063.38</td>
<td>41.13</td>
<td>4992.81</td>
<td>4804.56</td>
</tr>
<tr>
<td>80.ESC3.3</td>
<td>-</td>
<td>2052.40</td>
<td>37.47</td>
<td>5454.82</td>
<td>5250.07</td>
<td>62.38</td>
<td>4869.74</td>
<td>4577.13</td>
<td>55.44</td>
<td>3890.21</td>
<td>3722.78</td>
<td>45.08</td>
<td>5273.30</td>
<td>5138.51</td>
</tr>
<tr>
<td>80.ESC3.4</td>
<td>-</td>
<td>2885.00</td>
<td>58.43</td>
<td>5349.24</td>
<td>5029.86</td>
<td>51.02</td>
<td>5288.16</td>
<td>5141.49</td>
<td>54.66</td>
<td>4717.45</td>
<td>4479.56</td>
<td>54.33</td>
<td>5325.76</td>
<td>5183.78</td>
</tr>
<tr>
<td>80.ESC3.1</td>
<td>-</td>
<td>7802.60</td>
<td>73.22</td>
<td>16840.20</td>
<td>16690.60</td>
<td>433.78</td>
<td>16337.70</td>
<td>16073.80</td>
<td>180.42</td>
<td>14215.30</td>
<td>13951.50</td>
<td>55.38</td>
<td>16665.80</td>
<td>16131.80</td>
</tr>
<tr>
<td>80.ESC3.2</td>
<td>-</td>
<td>8020.20</td>
<td>63.61</td>
<td>16690.20</td>
<td>16350.40</td>
<td>211.89</td>
<td>16436.10</td>
<td>16148.90</td>
<td>125.65</td>
<td>14881.20</td>
<td>14521.80</td>
<td>74.41</td>
<td>16767.90</td>
<td>16436.60</td>
</tr>
<tr>
<td>80.ESC3.3</td>
<td>-</td>
<td>8507.00</td>
<td>78.20</td>
<td>16526.30</td>
<td>16112.20</td>
<td>94.65</td>
<td>16513.20</td>
<td>16299.70</td>
<td>98.21</td>
<td>15671.70</td>
<td>15432.20</td>
<td>80.16</td>
<td>16532.40</td>
<td>15800.80</td>
</tr>
<tr>
<td>80.ESC3.4</td>
<td>-</td>
<td>10706.00</td>
<td>77.68</td>
<td>16800.00</td>
<td>16379.00</td>
<td>60.65</td>
<td>16761.00</td>
<td>16508.60</td>
<td>68.78</td>
<td>16508.10</td>
<td>16380.10</td>
<td>65.68</td>
<td>16923.40</td>
<td>16711.30</td>
</tr>
<tr>
<td>80.rbg109a</td>
<td>-</td>
<td>207.60</td>
<td>141.17</td>
<td>465.14</td>
<td>446.47</td>
<td>321.16</td>
<td>386.34</td>
<td>370.78</td>
<td>196.82</td>
<td>359.42</td>
<td>355.71</td>
<td>123.96</td>
<td>447.08</td>
<td>428.53</td>
</tr>
<tr>
<td>80.rbg150a</td>
<td>-</td>
<td>350.00</td>
<td>254.84</td>
<td>596.08</td>
<td>589.87</td>
<td>571.31</td>
<td>549.06</td>
<td>526.21</td>
<td>209.67</td>
<td>540.71</td>
<td>534.84</td>
<td>128.29</td>
<td>571.87</td>
<td>563.19</td>
</tr>
<tr>
<td>80.rbg174a</td>
<td>-</td>
<td>406.60</td>
<td>292.49</td>
<td>710.25</td>
<td>692.64</td>
<td>&gt;600.00</td>
<td>652.84</td>
<td>642.07</td>
<td>435.92</td>
<td>625.55</td>
<td>619.34</td>
<td>246.19</td>
<td>698.90</td>
<td>686.82</td>
</tr>
</tbody>
</table>

Table A.6: The local search heuristics tested on the non-clustered problem instances (80% version).