Erasmus University Rotterdam

Master Thesis Operations Research & Quantitative Logistics

A Branch-and-Cut Algorithm for the Traveling Salesman Problem with Drone

Author:

Eveline van Dijck, 455264

Supervision:

Dr. P.C. Bouman, coach

Dr. R. Spliet, co-reader

August 27, 2018

Abstract

In this thesis, we introduce logical and comb inequalities for the Traveling Salesman Problem with assistance of a Drone, where a truck and drone work together in a coordinated tour that visits all customers. We have solved randomly generated instances with up to 40 customer nodes using a branch-and-cut algorithm implemented by CPLEX and assessed the performance of the proposed inequalities. Moreover, we present an extensive analysis to determine the benefit from the proposed methods by running two versions of the algorithm in parallel in an optimistic environment, where violated cuts are identified immediately.

Contents

List of Tables **iii**

List of Figures **iii**

1 Introduction **1**

2 Research gap **3**

3 Literature review **5**

3.1 Traveling Salesman Problem 5

3.2 Prize Collecting Traveling Salesman Problem 10

3.3 Other related variants of the TSP 12

3.4 Traveling Salesman Problem with assistance of a drone 14

3.5 Existing work on drone delivery problems 16

4 Valid inequalities for the TSP-D **18**

4.1 Initial blossom inequality 18

4.2 Alternative blossom inequality 19

4.3 Logical inequalities 21

5 Separation algorithms **22**

5.1 Separation of sub-tour inequalities 22

5.2 Separation of blossom and simple comb inequalities 23

6 Implementation of the branch-and-cut algorithm **30**

6.1 Branch-and-cut algorithm in CPLEX 30

6.2 Finding cuts 31

6.3 Experiment design 32

6.4 Instance generation 34

7 Computational results **35**

7.1 Preliminary results 35

7.2 Extensive results 45

8 Conclusion **51**

9 Further research **52**

10 Bibliography **53**

A Formulations **58**

A.1 TSP formulation 58

A.2 PCTSP formulation 59

B Numeric results **59**

B.1 Preliminary analysis 59

B.2 Extensive analysis 63

List of Tables

1 Exact methods for TSP 3
2 Solution methods for TSP-D 4
3 Tested versions 33
4 Solution times (minutes:seconds) using `Use Cut Force`, averages of 30 instances 36
5 Solution times (minutes:seconds) using `Use Cut Force`, averages of 30 instances 36
6 Number of nodes in the branch-and-cut tree N and number of user cuts added U, using $t = 0$ and `Use Cut Force`, averages of 30 instances 37
7 Average differences in N and p-values 38
8 Results using $n = 20$ and `Use Cut Force`, averages of 30 instances 42
9 Results using `Use Cut Purge`, $t = 0$, averages of 30 instances 43
10 Results preliminary analysis, averages of 30 instances (I) 60
11 Results preliminary analysis, averages of 30 instances (II) 61
12 Results preliminary analysis, averages of 30 instances (III) 62
13 Results for instance sizes in $[32, 38]$, $t = 0$, averages of 30 instances 63
14 Results comparing versions A and C, $n = 25$ and $t = 0$ (I) 63
15 Results comparing versions A and C, $n = 25$ and $t = 0$ (II) 64
16 Results comparing versions A and C, $n = 25$ and $t = 0$ (III) 65
17 Results comparing versions A and C, $n = 30$ and $t = 0$ (I) 65
18 Results comparing versions A and C, $n = 30$ and $t = 0$ (II) 66
19 Results comparing versions A and B, $n = 20$ and $t = 2$ (I) 67
20 Results comparing versions A and B, $n = 20$ and $t = 2$ (II) 68

List of Figures

1 Outline of the branch-and-cut algorithm 6
2 Example of a violated comb 8
3 Example of a feasible solution 8
4 Example of a blossom 9
5 Example of a simple comb 9

6 Example of a solution using only the truck 15
7 Example of a solution using only the drone 15
8 Example of a solution using truck and drone 15
9 Example of a comb in an arbitrary instance 28
10 Creating a variant by deleting nodes from handle 28
11 Creating a variant by including loose nodes 28
12 Creating a variant by swapping tooth nodes 28
13 Creating a variant by swapping teeth 28
14 Creating a variant by changing teeth 28
15 Example of a comb with even number of teeth 29
16 Creating a variant by adding an extra tooth 29
17 Number of comb constraints for $n = 30$, per instance. Instances where no comb constraints were generated are not shown. 39
18 Comparison of number of comb inequalities for different instance sizes, averages over 30 instances 40
19 The number instances with a smaller tree, $t = 0$. In total 30 instances were tested. Remaining instances had an equal N in both versions. 41
20 Averages over 30 instances, $t = 0$ 44
21 The number instances with a smaller tree, $t = 0$ 44
22 Number of nodes for all instances for $n = 25$ 46
23 Time and optimality gap for all instances when $t = 0$ and $n = 25$ 46
24 Number of nodes for all instances in version A and C 48
25 Time in branch-and-cut tree and optimality gap when the other version has solved the problem 48
26 Number of nodes for all instances with $t = 2$, $n = 20$ 49
27 Time and optimality gap for all instances when $t = 2$ and $n = 20$ 50

1 Introduction

Since its introduction in a seminar talk by Hassler Whitney in 1934 and its introduction in scientific literature by Flood (1955), the traveling salesman problem has been researched extensively. An overview of the history of the traveling salesman problem, the theoretical and computational results derived over the years, and an explanation of the most advanced algorithm that is used at this moment can be found in the book by Applegate et al. (2011). The classic traveling salesman problem, abbreviated as TSP in this thesis, is concerned with finding a single route with minimal costs through all nodes in a graph. Next to the obvious practical application of finding the least cost route through geographical locations, there are many more applications of the TSP, such as mapping the human genome and drilling problems (Ben-Dor and Chor, 1997; Lin and Kernighan, 1973). Algorithms for solving the TSP with geographical locations are widely used in planning software, resulting in a route for people, vehicles or parcel delivery. This thesis will focus on this last topic, as recent developments in drone technology give rise to a variant of the TSP where a parcel delivery truck is assisted by a drone. Providing a delivery truck with a drone gives many benefits as it has "best of both worlds" (Agatz et al., 2018). The main advantages of a drone are its speed, its independence of traffic, and its low costs, as it is not operated by a human pilot. However, both the load capacity and the travel range of a drone are limited, which implies that drone delivery is mostly useful for last-mile delivery. Parcel delivery by drones is already being tested by postal companies all over the world and is supported by the developments in delivery services and e-commerce (UnmannedCargo, 2016). At this moment, 86% of Amazon's parcels in the USA weighs under 5 pounds, making it ideal for drones to carry, and the company is making great progress in drone delivery (Wang, 2015; Oswald, 2017). Amazon drones are able to deliver packages within 30 minutes for one dollar and 82% of customers are willing to pay for drone delivery service, making drone delivery very efficient regarding both the costs and time needed for delivery (Keeney, 2015; Snow, 2014). In 2017, UPS reported to have successfully performed a drone operation consisting of launching the drone from the top of a delivery van, delivering a package to a customer and returning to the van, while the delivery van visits another customer (UPS, 2017). Nowadays, frequent usage

of delivery drones still faces many economic, regulatory and technological obstacles. But as delivering with truck and drone can be very advantageous from an economic point of view, there will certainly be massive disruptions and new business models emerging in the field of drone delivery (Lee et al., 2016; Campbell et al., 2017).

The variant of the traveling salesman problem with assistance of a drone (TSP-D) poses many changes with respect to the classic traveling salesman problem. In the TSP-D the truck drives a tour while visiting customer nodes. The truck is equipped with a drone, which can be launched while the truck is parked at a customer location. From that moment on, the truck and drone are performing tasks in parallel. It is assumed that the drone can only carry one parcel at a time due to its limited capacity, and therefore it can only visit one customer per flight operation. After delivering the parcel, the drone and truck will reunite at any customer node. In the meantime, the truck can visit one or more customer nodes, or stay at the same customer waiting for the drone to return. In the traveling salesman problem with drone, the objective is to find a tour visiting all customers with minimal costs. These costs can be expressed in terms of distance, travel time, fuel and electricity costs, or environmental impact, but we will use travel time in this thesis. Moreover, it is assumed there is one depot where the truck starts and ends its tour. One could solve an instance without depot by repeatedly solving this problem for all customers as being the depot. Furthermore, it is assumed that all customers can be visited by the truck and the drone. In case a customer can only be visited by one of the two systems, this could be incorporated easily in the problem, making the problem easier to solve. The goal of this thesis is to contribute to contemporary research by finding valid inequalities for the TSP-D and using them efficiently in a branch-and-cut algorithm, as is explained in Section 2. Section 3 presents an overview of available research to the TSP-D and related combinatorial problems. Valid inequalities for the TSP-D are derived in Section 4 and Section 5 discusses the way they are found in a branch-and-cut algorithm. All implementation details of this algorithm are noted in Section 6, followed by results of computational experiments in Section 7. Finally, the conclusion and suggestions for future research will be covered in Section 8 and Section 9, respectively.

2 Research gap

As drone-assisted parcel delivery has become technically possible only recently, research to the TSP-D is not as extensive as research to the TSP, which has been around for decades. As the research in this thesis focuses on solving the TSP-D with an exact algorithm, we focus on exact solving methods. The most straightforward method for solving the TSP is listing all permutations and choosing the cheapest tour in that list. The dynamic programming approach developed by Held and Karp (1962) is also exact and has the best time-complexity for any known algorithm capable of solving all instances of the TSP. The research to the TSP has been greatly influenced by the work of Dantzig et al. (1954), who introduced the cutting plane method. Ideas from this method are used in the branch-and-cut algorithm, which is the best performing exact algorithm at this moment. This is mainly due to the fact that solving the subproblems can be done quite fast and the branch-and-cut algorithm guarantees that the found solution is optimal. All exact methods are summarized in Table 1.

Table 1: Exact methods for TSP

Procedure	Running time	Max nr of cities
Find cheapest tour among all permutations	$O(n!)$	
Held-Karp algorithm (Held and Karp, 1962)	$O(n^2 2^n)$	
Branch-and-bound algorithms		40-60
Cutting plane method		15 112
Branch-and-cut algorithms		85 900

For the TSP-D, only few articles have been published. The first article to introduce the problem was written by Murray and Chu (2015). A mixed integer program was formulated and simple heuristics were constructed and tested. Ha et al. (2015) propose two other heuristic and reformulate the problem. This reformulation is comparable to the mixed integer programming formulation used by Agatz et al. (2018). They also propose methods to construct a TSP-D tour from an optimal TSP tour, based on local search and dynamic programming. Methods using meta-heuristics were developed by Ferrandez et al. (2016), Ponza (2016) and Freitas and Penna (2018), who developed genetic algorithms, simulated annealing methods

and a variable neighborhood search, respectively. An exact dynamic programming approach was established by Bouman et al. (2017). In this paper, instances with 16 customer nodes are solved to optimality. When limiting the number of nodes the truck can visit on its own, instances with up to 20 customer nodes are solved to optimality. Finding the optimal solution for those instances, allowing the truck to visit two customers on its own, yields an average computation time of more than 10 hours. When visits by trucks only are disabled for an instance of 20 customer nodes, computing the optimal solution takes a little over 27 minutes. Table 2 gives a full overview of the available methods.

Table 2: Solution methods for TSP-D

	Procedure	Source
Exact algorithms	Integer linear programming	Agatz et al. (2018)
	Dynamic Programming	Bouman et al. (2017)
Heuristic methods	Simple heuristics	Murray and Chu (2015), Ha et al. (2015)
	Genetic algorithm	Ferrandez et al. (2016)
	K-means algorithm	Ferrandez et al. (2016)
	Simulated annealing	Ponza (2016)
	Variable neighborhood search	Freitas and Penna (2018)
Approximation algorithms	Improvements from TSP tour	Agatz et al. (2018)

Note: Based on a similar table in Bouman et al. (2017)

It is observed that exact methods that are well performing for the TSP have not been developed for the TSP-D. As the TSP-D differs substantially from the TSP and other existing variants, one cannot use theoretical results from the TSP directly in the TSP-D. This thesis is concerned with solving the TSP-D with a branch-and-cut method, which requires derivations of templates of cuts and identification of violations of cuts. The main difference between the TSP-D and the classic TSP is that the truck does not have to visit all nodes in the graph and has to be coordinated with a second system, which is a drone in this case. This surely complicates the derivations of cuts. Deriving templates for valid inequalities and constructing identification algorithms to find them for the TSP-D are the main goals of this thesis. The main focus is on comb inequalities, which were introduced by Hong (1972). These inequalities

prove to be useful in solving the more difficult instances of TSP-like problems. (Applegate et al., 2011)

3 Literature review

The articles and other sources assessed for the literature review concern the classic Traveling Salesman Problem, the Prize Collecting Traveling Salesman Problem, other related TSP variants, the Traveling Salesman Problem with Drone, and other combinatorial problems with drones reviewed in that order.

3.1 Traveling Salesman Problem

As mentioned earlier, the Traveling Salesman Problem is concerned with visiting all nodes in a graph with minimal costs. The TSP generalizes most of the problems considered in this thesis. The used formulation for the TSP in Appendix A.1 was introduced by Dantzig et al. (1954). The TSP is modeled as a set V of nodes and a set E of edges. Together these sets form an undirected graph $G = (V, E)$. In this formulation, decision variable x_{ij} defines for each edge $(i, j) \in E$ if it is used in the optimal tour. Parameter c_{ij} is the cost of using edge (i, j). It is assumed that c_{ij} is defined for all $(i, j) \in E$, and therefore, traveling is possible between all nodes $v \in V$. The objective is to minimize the sum of the costs of all edges that are used in the tour. The first constraint ensures that all nodes in the set V are visited by the traveling salesman and the second constraint makes sure all variables are either 0 or 1.

3.1.1 Branch-and-cut algorithm for the Traveling Salesman Problem

As explained in Section 2, the branch-and-cut method is the most successful method in solving the TSP optimally. Because of its importance for this thesis, we will discuss it in detail in this section. The branch-and-cut algorithm is a combination of the cutting plane method and the widely known branch-and-bound algorithm. The cutting plane method for the TSP was introduced by Dantzig et al. (1954). The method solves the linear programming relaxation of a problem iteratively, and adds cuts after each iteration. With each cut, the feasible region of the linear programming relaxation is shrunk without deleting possible solution tours. An

inequality that cuts the solution space of the linear programming relaxation, but does not cut any feasible solutions to the original integer programming problem, is called a *valid* inequality, or *facet-defining*. The procedure of solving the linear programming relaxation and searching for valid cuts is repeated until a feasible solution for the original problem is obtained. In the branch-and-cut algorithm, the first step is to initialize a linear programming relaxation of the original problem. Initially, a cutting plane procedure is used until no more valid inequalities can be found anymore. The best solution for the original problem and the relaxed problem is stored. Then, the first branching step is taken on a fractional variable, which means this fractional variable is restricted to be either 0 or 1. This yields two new nodes in the so-called branch-and-cut tree. In every node, the new linear programming relaxation of a problem is solved. If the solution of the relaxed problem is higher than the best solution found for the original problem, this means there is no room for improvement in this branch of the tree and the node is *pruned*, i.e. cut off. Otherwise, the procedure of branching, solving and looking for valid inequalities is repeated. Whenever a better incumbent solution is found, the bounds throughout the tree are updated. The procedure of the branch-and-cut algorithm is summarized in Figure 1. The implementation of this algorithm in this thesis is explained in detail in Section 6.

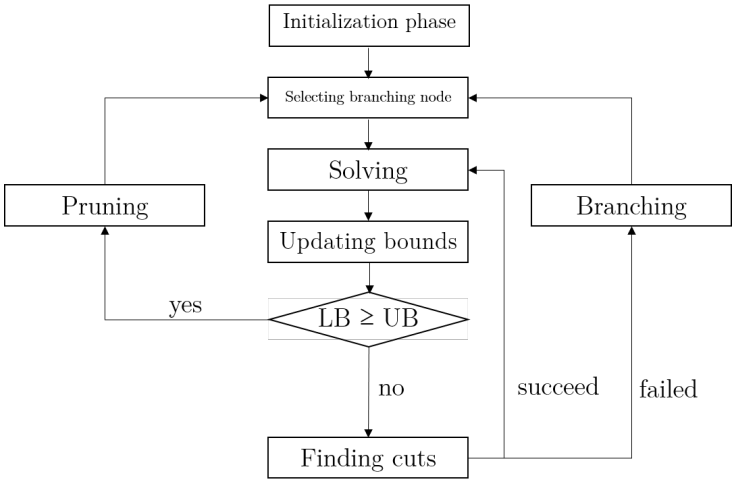


Figure 1: Outline of the branch-and-cut algorithm

The search for valid inequalities can be either exact or heuristic. *Exact separation procedures* guarantee to find all violations of a certain type of inequalities. *Heuristic separation proce-*

dures aim to find violated inequalities, but do not guarantee to find violations. Therefore, the exact separation procedures generally cut off more of the solution space, yielding a tighter formulation. A tighter formulation might cause a decrease in the number of nodes needed to arrive at the optimal solution. The main advantage of heuristic separation procedures is their speed, which is why they are often used in state-of-the-art branch-and-cut algorithms.

3.1.2 Templates of cuts in the TSP

Sub-tour inequalities were introduced in Dantzig et al. (1954) and are necessary for a complete formulation of the TSP. They ensure that for every subset of nodes, there are at least 2 paths going in and out the subset. There can also be multiple paths through the subset, but the number of used edges surpassing the border of the subset should always be even.

$$\sum(x_{ij} : (i,j) \text{ has one end in } S \text{ and one end not in } S) \geq 2 \text{ for all } S \subset V, S \neq \emptyset \quad (1)$$

Comb inequalities as in Equation 2 were introduced by Hong (1972) and use subsets shaped like a comb, where H is the handle of the comb and T_j is tooth $j \in \{1, \ldots, s\}$. Combs are officially defined by Equations 3-6.

$$\sum_{(i,j)\in E:|(i,j)\cup H|=1} x_{ij} + \sum_{j=1}^{s} \sum_{(i,j)\in E:|(i,j)\cup T_j|=1} x_{ij} \geq 3s + 1 \quad (2)$$

$$|H \cap T_j| \geq 1 \qquad \forall j = 1, \ldots, s \quad (3)$$

$$|T_j \setminus H| \geq 1 \qquad \forall j = 1, \ldots, s \quad (4)$$

$$|T_i \cap T_j| = 0 \qquad 1 \leq i < j \leq s \quad (5)$$

$$s \text{ odd} \quad (6)$$

To illustrate the working of this inequality, we consider the components of the comb inequality separately in the case where all teeth have a single path going through them, yielding a minimal left-hand side of the inequality. The first term sums the weight of all edges surpassing the border of the handle. As every tooth has at least one node within the handle and at least one outside of it because of Equations 3 and 4, the single path through a tooth is an edge surpassing the border of the handle. Extending this to all teeth, the term $\sum_{(i,j)\in E:|(i,j)\cup H|=1} x_{ij}$

would thus yield a minimum of s. The second term of the inequality sums the edges surpassing the border of the teeth. In case there is a single path going through each tooth, there must be two edges surpassing its border. This yields a minimum value of $2s$ for the term $\sum_{j=1}^{s} \sum_{(i,j)\in E:|(i,j)\cup T_j|=1} x_{ij}$. Therefore, the left-hand side of Equation 2 would always be greater than or equal to $3s$. However, as s is by definition odd, $3s$ is odd as well. As the number of nodes going in and out of a subset is supposed to be even, the right-hand side can be increased by one in any case. This results in the comb inequality above. Figure 2 shows a fractional solution that is not a valid tour, but all sub-tour constraints are met. The dotted edges in this figure have a weight of 0.5, the others a weight of 1. Therefore, the left hand side of the equation is equal to 9, as the sums of edge weights crossing the borders of the handle and the teeth are 3 and 6, respectively, and the right hand side is 10. For the comb in Figure 3, both the left and right hand side are equal to 10 and therefore, this solution does not violate the comb inequality. In case each tooth only contains 2 nodes, the comb inequality is also called a *2-matching inequality* or *blossom inequality* in literature. When the intersection of the handle with each tooth consists of one single node, it is called a *simple comb*. Figures 4 and 5 show examples of both types of inequalities. As separation algorithms are mainly available for these types, the focus of this thesis is on finding blossom and simple comb inequalities.

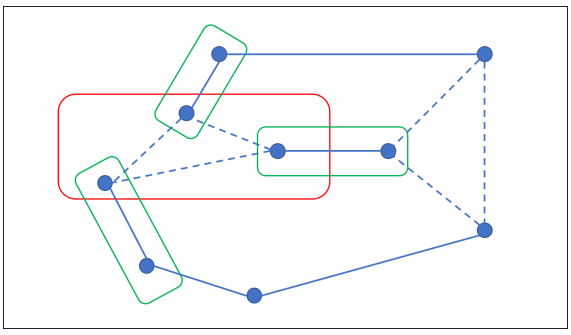


Figure 2: Example of a violated comb

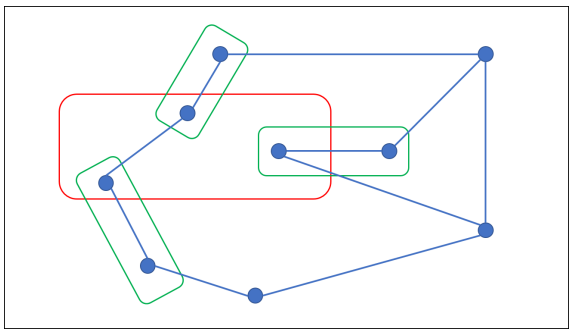


Figure 3: Example of a feasible solution

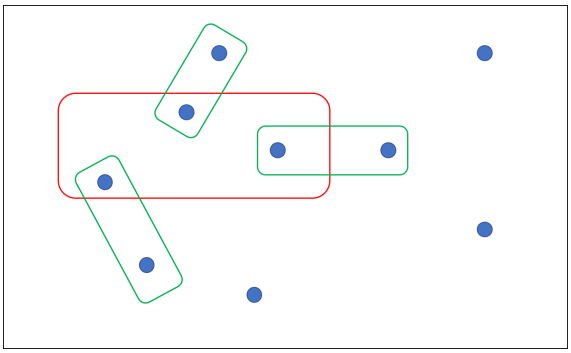


Figure 4: Example of a blossom

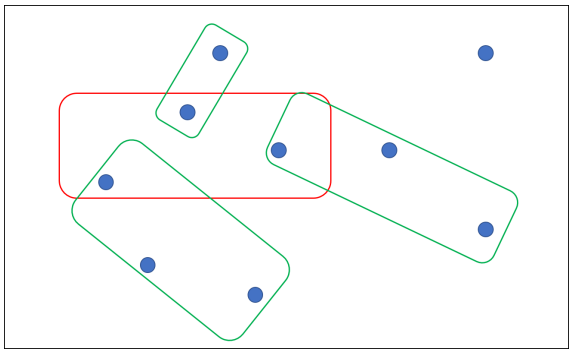


Figure 5: Example of a simple comb

In the TSP, comb inequalities are generalized by clique tree inequalities and path inequalities. Moreover, in the state-of-the-art branch-and-cut algorithm valid inequalities are found by multiple separation algorithms, shrinking techniques and cut metamorphoses. A full overview of these approaches can be found in Applegate et al. (2011). These cuts and techniques for the TSP-D are not considered in this thesis, but can be investigated in further research.

3.1.3 Finding cuts in the TSP

In 1993, the branch-and-cut algorithm was able to solve previously unsolved cases of the TSP by implementing several separation algorithms. (Applegate et al., 1993) The shrinking procedure was first formalized by Padberg and Rinaldi in 1990, but was already used by Crowder and Padberg in 1980 (Padberg and Rinaldi, 1990b; Crowder and Padberg, 1980). The idea behind shrinking is that the shrunk node set is smaller, and the violated cuts are more easily visible. Shrinking is often used as a pre-processing step (Applegate et al., 2011). Padberg and Rinaldi (1990a) also developed a separation algorithm for sub-tour constraints. This algorithm solves $n - 1$ max-flow problems, but reduces the size of the problem by shrinking nodes on the same side of a minimum capacity cut. For blossom constraints, which are defined as comb equalities with only two nodes in each tooth, Padberg and Rao (1982) developed a polynomial time exact separation algorithm, solving an odd minimum cut-set problem. Calculations are sped up by using the shrinking procedure and considering connected components. For comb inequalities, two heuristic algorithms were developed. Grotschel and Holland (1991) shrink the solution graph such that some violated comb constraints are turned into violated blossom constraints and apply the Padberg-Rao

procedure afterwards. Padberg and Rinaldi (1990b) propose an algorithm that is based on a block decomposition of the graph and the concept of virtual edges. As both algorithms for finding combs are a heuristic, they might not identify existing violated cuts. Padberg and Rinaldi (1991) introduced, among other methods, the procedure of removing constraints during the branch-and-cut algorithm if they have not been relevant for a number of iterations. The removed constraints are then put in a cut pool, which makes them easier to identify if violated later. In the meantime, more advanced algorithms have been incorporated in the branch-and-cut method, such as the consecutive ones procedure, gluing and tightening. (Applegate et al., 2011) Again, these seem to be very interesting for future research.

3.2 Prize Collecting Traveling Salesman Problem

The Prize Collecting Traveling Salesman Problem (PCTSP) was introduced by Balas (1989). It concerns a traveling salesman problem where the salesman is penalized for every node that is not included in the tour. The total number of penalties cannot exceed a certain threshold, while traveling costs are minimized. The PCTSP is related to the traveling salesman problem with assistance of a drone in the sense that nodes can be skipped in both problems. The PCTSP formulation in Appendix A.2 was introduced by Balas (1995). In the PCTSP, the traveling salesman gets a prize w_i for every node i he visits. The goal is to find a minimal cost tour while collecting a minimum amount of w_0 of prize money. Decision variable x_{ij} is used to indicate whether arc (i, j) is used in the tour and y_i indicates whether node i is skipped, i.e. not included in the tour. Costs for using arcs and skipping nodes are defined on parameters c_{ij} and c_i, respectively. U is defined as $U = \sum_{i \in N} w_i - w_0$. Moreover, $G_L(x, y)$ is the subgraph of directed graph $G_L = (N, A \cup L)$ where $x_{ij} = 1$ and $y_i = 1$. The PCTSP is also researched without the penalty threshold. Bienstock et al. (1993) extended this problem to a Steiner Tree variant. For the PCTSP without the penalty threshold, lower bounds were derived and computational results were reported in Dell'Amico et al. (1995). The Steiner Tree variant of the TSP-D and the derivation of lower bounds independent of the drone's speed form also a gap in the research to the TSP-D.

3.2.1 Templates of cuts in the PCTSP

In early research to this problem, Balas has derived many polyhedral cuts for the PCTSP. (Balas, 1989, 1995). Balas (1989) derived sub-tour constraints for less complex polytopes in his search for PCTSP sub-tour constraints. For this research, the results for a polytope without the knapsack constraint are also of interest, as the TSP-D will, in general, not contain such a constraint. For simplicity, let $x(S,T) = \sum_{i \in S} \sum_{j \in T \setminus \{i\}} x_{ij}$ and $y(S) = \sum_{i \in S} y_i$. For this simplified variant of the PCTSP, Balas derives the valid sub-tour constraint in Equation 7 for all $S \subset V$, $2 \leq |S| \leq n-1$, and all $k \in S$, $l \in V \setminus S$. The sub-tour constraints for the PCTSP polytope can be strengthened, depending on the price parameters of the considered subset. In all cases, the constraints are facet-defining if $|S| \leq n-2$.

$$x(S,S) + y(S \setminus \{k\}) - y_l \leq |S| - 1 \tag{7}$$

Defining combs and comb inequalities is more difficult when feasible tours do not visit all nodes in a graph. Balas (1995) derived comb inequalities as in Equation 8. This inequality is facet-defining for both the PCTSP polytope and its earlier discussed simplified variant.

$$x(H,H) + \sum_{j=1}^{s} x(T_j, T_j) + y(H) \leq |H| + \frac{s-1}{2} \tag{8}$$

Source-destination inequalities were introduced for the asymmetric traveling salesman polytope by Balas and Fischetti (1993). This class of inequalities generalizes many other families of facets, one of which is the family of comb inequalities. As there are currently no separation algorithms available for source-destination inequalities, they will not be investigated in this thesis. In an implementation of a branch-and-cut algorithm for the PCTSP, Bérubé et al. (2009) also use inequalities from the associated knapsack polytope and logical inequalities. The latter type uses the fact that an edge can only be part of a solution if the adjacent edges are not in a loop. The inequalities in Equations 9 and 10 follow from this reasoning.

$$x_{ij} \leq 1 - y_i \qquad \forall (i,j) \in \delta(i), i \in V \setminus \{v_0\} \tag{9}$$

$$x_{ij} \leq 1 - y_j \qquad \forall (i,j) \in \delta(j), j \in V \setminus \{v_0\} \tag{10}$$

3.2.2 Solving the PCTSP

Feillet et al. (2005) consider three variants of the traveling salesman with profits, one of which is the PCTSP. In all variants is it possible to exclude customers from the tour, which is why the proposed solution approaches are of interest for this thesis. Before 1998, all solution approaches were based on branch-and-bound. Thereafter, two papers solved a symmetric TSP with profits with the branch-and-cut algorithm, yielding optimal solutions for 300 and 500 nodes, respectively. Later, Bérubé et al. (2009) used a branch-and-cut algorithm solving the PCTSP. Using the results of Balas (1995), they defined blossoms and simple combs and separated these cuts with the heuristic procedures of Padberg and Rinaldi (1990b) and Fischetti et al. (1998). Their algorithm also includes column generation. From this report it is concluded that slightly more instances can be solved when using comb inequalities and that they function best for TSP instances, compared to instances of the Vehicle Routing Problem.

3.3 Other related variants of the TSP

In this section, available research on other problems with similarities to the TSP-D is summarized. The Orienteering Problem (OP) is closely related to the PCTSP, as prizes are collected when visiting nodes and tours that skip nodes are feasible. The objective of the OP is to find a tour that maximizes the total amount of collected prizes while a time threshold is not exceeded. (Feillet et al., 2005) Polyhedral results for the OP are similar to the results for the PCTSP and are therefore not discussed in this literature review. Fischetti et al. (1998) describe a branch-and-cut and price algorithm for the OP, as well as separation algorithms for comb inequalities. Violated comb inequalities are identified by inspecting the connectivity of minimum-weight spanning trees found by the greedy algorithm of Kruskal (1956). In Gendreau et al. (1998), a branch-and-cut algorithm is developed for the OP, where comb inequalities are found by the Padberg and Rinaldi (1990b) algorithm. A remarkable aspect to their solution method is that both the number of constraints identified and the number of constraints generated is limited by 50 and 30, respectively. Moreover, heuristics are used within the branch-and-cut algorithm to define a feasible solution bound for a sub-

problem. Finally, branching is done primarily on the y_k variables of the formulation of the STSP, meaning they branch on the decision to include a node in the tour. If all variables y_k are integer, a similar procedure is followed for x_{ij}. The preference for branching on y_k variables is also used in this thesis. Both Gendreau et al. (1998) and Fischetti et al. (1998) also use logical inequalities in their branch-and-cut algorithm. Their usage corresponds with the usage for the PCTSP. Finally, Gendreau et al. (1998) and Fischetti et al. (1998) solve instances with up to 300 and 500 nodes to optimality, respectively. Next to the PCTSP and OP, another non-spanning variant of the classic TSP is the Generalized Traveling Salesman Problem (GTSP). In this problem, the nodes are clustered and the objective is to find a minimal cost tour where each cluster is visited at least once. A complete undirected graph $G = (V, E)$ is given. Node subsets $C_1, \ldots, C_m$ form a proper partition of V and are called clusters. c_{ij} is the cost of using edge (i, j). Decision variable x_{ij} is 1 if edge (i, j) is selected in the considered tour, y_i is a variable indicating whether node i is visited. Inequalities for the GTSP were summarized in Fischetti et al. (1997). In that paper, the comb inequality in Equation 11 was introduced. In this equation, β_i is a parameter depending on how many clusters are contained in both the teeth and the handle and how many are contained in a tooth only.

$$\sum_{(i,j)\in E(H)} x_{ij} + \sum_{j=1}^{s} \sum_{(i,k)\in E(T_j)} x_{ik} + \sum_{i\in V} \beta_i(1 - y_i) \leq |H| + \sum_{j=1}^{s}(|T_j| - 1) - \frac{s+1}{2} \qquad (11)$$

When assuming there are no whole clusters in the TSP-D, this comb inequality is equal to the comb inequality for the PCTSP. Fischetti et al. (1997) were the first to develop a branch-and-cut algorithm for the GTSP. They propose several methods, both exact and heuristic, to find violated cuts in the symmetric variant of the GTSP. For comb inequalities, they use a heuristic method based on a procedure by Padberg and Grotschel (1985) for the TSP, based on the connectivity of the fractional graph. Another heuristic separation algorithm for finding combs is concerned with shrinking all clusters to a single node. This method is not considered in this research as there are, in general, no clusters in the TSP-D. Using the branch-and-cut algorithm, Fischetti et al. (1997) find optimal solutions for GTSP instances up to 442 nodes. The Covering Salesman Problem (CSP) is related to the TSP-D in the sense that the salesman does not have to visit all nodes in the graph. However, the nodes

that are not included in the tour should be within a certain distance of a visited node. The Covering Tour Problem (CTP) is very similar to the CSP but defines a set of nodes that must be included in the tour. The CSP, the CTP and the GTSP are all special cases of the Generalized Covering Salesman Problem, introduced by Golden et al. (2012). For the CTP, Gendreau et al. (1997) developed a branch-and-cut algorithm. They have developed several families of cuts, one of which is the family of blossom inequalities. Other derived inequalities include dominance constraints and sub-tour elimination constraints. In the proposed branch-and-cut algorithm, and in this thesis, simple combs are found with the algorithm by Padberg and Rinaldi (1990b). Gendreau et al. (1997) find optimal solutions for CTP for up to 600 nodes. The truck and trailer routing problem (TTRP) relates to the TSP-D in the sense that not all nodes are covered by the same systems. In the TTRP, a truck and trailer need to visit all customer nodes in a graph. Some customers are only accessible by the truck, meaning the truck has to leave the trailer waiting at a node in the main tour and visiting other nodes in the meantime via a sub-tour. A variant of this problem with a single truck and trailer and parking locations for the trailer outside customer nodes is called the Single Truck and Trailer Routing Problem with Satellite Depots and was solved with a branch-and-cut algorithm by Belenguer et al. (2016). Two variants of comb inequalities were used, one where some depots are in the handle of the comb and one where every tooth in the comb contains at least one depot. Using this approach, Belenguer et al. (2016) solved several instances with 100 customer nodes and up to 20 satellite depots.

3.4 Traveling Salesman Problem with assistance of a drone

This section introduces the TSP-D and its notation formally. In literature, the TSP-D was formulated in two different manners. Murray and Chu (2015) introduced the problem as the Flying Sidekick Traveling Salesman Problem (FSTSP), and used a very extensive formulation taking many aspects into account, such as the endurance of the drone. Agatz et al. (2018) and Ha et al. (2015) use a formulation that resembles the formulation in Appendix A.1 relaxing some of the constraints posed in the FSTSP. The latter uses other input that increases the number of variables and constraints, but considering the similarities between the TSP and TSP-D formulation, the formulation by Agatz et al. (2018) is used in this research. Again, a

graph $G = (V, E)$ is defined. For the traveling salesman problem with assistance of a drone, two type of costs are defined in terms of travel time. The travel time of the truck between node i and j is defined as $c(e) = c(v_i, v_j)$. Similarly, the travel time of the drone is defined as $c^d(e) = c^d(v_i, v_j)$. Nodes are divided in drone nodes, which are only visited by the drone, truck nodes, which are only visited by the truck, and combined nodes, visited by both systems. Next, the concept of an operation is introduced. An operation o consists of a combined start and end node, at most one drone node, and a non-negative number of truck nodes. The set of feasible operations is then defined as O. Parameter c_o is defined as the cost of an operation $o \in O$. Decision variable x_o indicates for each $o \in O$ if it is selected in the optimal tour. Finally, an auxiliary variable y_i is defined for each node $i \in V$ to indicate whether node i is used as a start node in the solution. Several subsets of O have been defined. $O^-(i)$ and $O^+(i)$ contain all operations o that, respectively, start and end in node i. $O(ij)$ contains all operations starting in node i and ending in j. $O(i)$ is the set of all operations o that contain node i. $O^-(S)$ and $O^+(S)$ are subsets containing all operations that start in subset S and end in $V \setminus S$ and end in S and start in $V \setminus S$, respectively. Figure 8 depicts a feasible solution for the TSP-D using 4 operations. Drone nodes and truck nodes are represented as yellow and green points, respectively. Combined nodes are blue. Important to note is that in this thesis, only operations with at most one truck node are considered. Comparing this solution with the truck only solution in Figure 6 and the drone only solution in Figure 7 it is clear that the coordination of truck and drone yields a more complex solution structure, but, as it has been shown by Agatz et al. (2018), it can reduce the delivery time substantially.

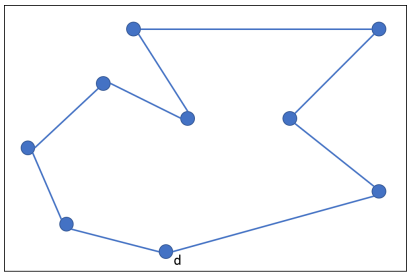


Figure 6: Example of a solution using only the truck

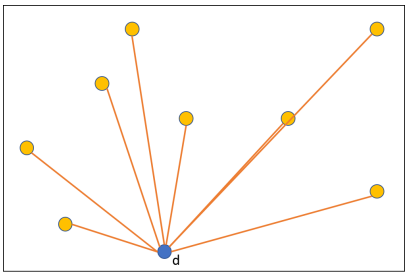


Figure 7: Example of a solution using only the drone

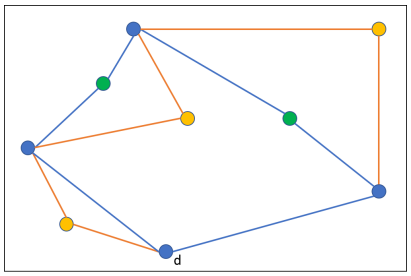


Figure 8: Example of a solution using truck and drone

Agatz et al. (2018) developed the integer programming formulation of the TSP-D in Equations 12-20. For all assumptions used in this formulation, the reader is referred to Agatz et al. (2018). For this operation oriented formulation, a sub-tour constraint was included in Equation 16 in the formulation.

$$\min \quad \sum_{o \in O} c_o x_o \tag{12}$$

$$\text{s.t.} \quad \sum_{o \in O(i)} x_o \geq 1 \qquad \forall i \in V \tag{13}$$

$$\sum_{o \in O^+(i)} x_o \leq n \cdot y_i \qquad \forall i \in V \tag{14}$$

$$\sum_{o \in O^+(i)} x_o = \sum_{o \in O^-(i)} x_o \qquad \forall i \in V \tag{15}$$

$$\sum_{o \in O^+(S)} x_o \geq y_i \qquad \forall S \subset V \setminus \{v_0\}, i \in S \tag{16}$$

$$\sum_{o \in O^+(v_0)} x_o \geq 1 \tag{17}$$

$$y_{v_0} = 1 \tag{18}$$

$$x_o \in \{0, 1\} \qquad \forall o \in O \tag{19}$$

$$y_i \in \{0, 1\} \qquad \forall i \in V \tag{20}$$

3.5 Existing work on drone delivery problems

Whereas Section 2 summarizes available literature on solving the TSP-D with state of the art techniques, other research focuses on relaxing the assumptions made by Agatz et al. (2018) or extending the problem in a different way. Marinelli et al. (2017) consider a version of the TSP-D where the truck and drone can meet *en route* instead of at a customer node. To solve this variant, they use the Greedy Randomized Adaptive Search Procedure, which was earlier used by Ha et al. (2015). The TSP-D was researched from an environmental perspective by Park et al. (2018), who conclude that the global warming potential (GWP) per 1 km delivery by a motorcycle is six times as high as the GWP when using a drone, and the amount of particulates produced by motorcycle delivery was twice the size of the

amount produced by drone delivery. The cooperation of trucks with drones in a delivery system is also researched in other set-ups. The Heterogeneous Delivery Problem (HDP) and the Multiple Warehouse Delivery Problem (MWDP), both introduced by Mathew et al. (2015), are Vehicle Routing Problems using drones as a delivery tool. In the HDP, a truck is equipped with a drone and locations where the truck can stop safely are marked as street vertices. The drone is then only used for last mile delivery to customers. An optimal solution is obtained by reducing the HDP to a Generalized Traveling Salesman Problem, a variant of the TSP discussed earlier in this thesis. In the MWDP, the street vertices are replaced by static warehouses and a fleet of drones is responsible for all deliveries. This problem is solved by reducing it to a Traveling Salesman Problem and by an exact polynomial time algorithm. The Same-Day Delivery Routing Problem with Heterogeneous Fleets, introduced by Ulmer and Thomas (2017), uses multiple trucks and drones to yield same-day delivery. They solve this problem with a Markov decision process model and Approximate Dynamic Programming. Another problem where multiple trucks and drones are used to deliver parcels is the Vehicle Routing Problem with Drones (VRPD). A worst case analysis for this problem was done by Wang et al. (2017), and the most advantageous truck and drone combinations were found by Poikonen et al. (2017). Carlsson and Song (2017) approach this problem using the continuous approximation paradigm, and find that the improved efficiency depends on the relative speeds of the vehicles and the number of drones used. For the variant of the VRPD with time windows, Di Puglia Pugliese and Guerriero (2017) introduced a mathematical model and solved several instances with CPLEX. A variant of the VRPD where drones are not linked to a single truck was introduced and solved with a local search heuristic by Daknama and Kraus (2017). Dorling et al. (2017) introduce two Vehicle Routing Problems using a single drone to deliver parcels and use a simulated annealing algorithm to solve them. Moreover, Dorling et al. (2017) introduces a cost function considering energy consumption and re-usage of drones. Variations on the drone-only delivery problem were researched by Vorotnikov et al. (2017) and Othman et al. (2017). A multi-trip version of the VRPD with exact energy calculation was solved by Cheng et al. (2018) using a branch-and-cut algorithm. Other related problems include the multi-objective Green UAV Routing Problem by Coelho et al. (2017) and drone delivery in health care by Scott and Scott (2017). A more extensive

overview of research on the drone delivery problems mentioned can be found in Freitas and Penna (2018). A full literature review on drones in logistic settings can be found in Otto et al. (2018).

4 Valid inequalities for the TSP-D

In this section, several valid inequalities are developed for the Traveling Salesman Problem with assistance of a Drone. As in the similar combinatorial problems covered in the previous section, we will derive comb inequalities and logical inequalities. The combs are based on the literature described in Sections 3.1 and 3.2. The logical inequalities are derived from the branch-and-cut algorithms for the problems described in Section 3.3.

4.1 Initial blossom inequality

An initial idea for defining comb inequalities is:

$$\sum_{o \in O^-(H)} x_o + \sum_{o \in O^+(H)} x_o + \sum_{j=1}^{s} \left(\sum_{o \in O^-(T_j)} x_o + \sum_{o \in O^+(T_j)} x_o \right) \geq 3s + 1 \tag{21}$$

The idea behind this inequality is that the operations are considered as arcs from their start node to their end node, and the chosen operations O' will span a directed Eulerian graph G'. This inequality, however, allows for teeth and a handle containing only nodes where no operations start or end and is therefore not valid. To have valid inequalities, it should be guaranteed that every subset $T_j \setminus H$ and $H \cap T_j$ contains at least one node that is the start node of an operation, i.e. $y_v = 1$. To incorporate this, the following constraints for a blossom inequality were drafted.

$$\sum_{i \in T_j} y_i - 1 \leq z_j \qquad \forall j = 1, \ldots, s \tag{22}$$

$$z_j \leq y_i \qquad \forall i \in T_j, \forall j = 1, \ldots, s \tag{23}$$

$$\sum_{j=1}^{s} z_j - s \leq \mathbf{z} - 1 \tag{24}$$

$$\sum_{j=1}^{s} z_j - s \geq -s \cdot (1 - \mathbf{z}) \tag{25}$$

$$z_j \in \{0,1\} \qquad \forall j = 1,\ldots,s \tag{26}$$

$$\mathbf{z} \in \{0,1\} \tag{27}$$

These constraints will not cut the feasible region of the problem as they are only used to define new variables z_j and $\mathbf{z}$. The binary variable z_j is equal to 1 if nodes k, l in subset T_j have $y_k = 1$ and $y_l = 1$, and is zero otherwise. Binary variable $\mathbf{z}$ is 1 if all z_j are equal to 1. Using $\mathbf{z}$ as an indicator variable, the new comb inequality becomes:

$$\sum_{o \in O^-(H) \cup O^+(H)} x_o + \sum_{j=1}^{s} \sum_{o \in O^-(T_j) \cup O^+(T_j)} x_o \geq \mathbf{z} \cdot (3s+1) \tag{28}$$

The main advantage of this formulation is that it is derived from the comb inequalities for the TSP and, therefore, is a valid inequality for the TSP-D. An obvious disadvantage of the derived blossom constraints is that variables z and $\mathbf{z}$ have to be added to the MIP formulation of the TSP-D which is used by the branch-and-cut algorithm. This is generally very hard to model. Moreover, many constraints are added to the formulation that may not be necessary in later steps of the cutting plane or branch-and-bound algorithm. As an alternative formulation of this comb inequality, which does not need additional variables in the main MIP, is presented in the next section, we will not use the comb inequality in Equation 28 in our computational experiments.

4.2 Alternative blossom inequality

In the researched literature, we have seen that comb inequalities could be defined for both directed and undirected graphs. Originally, the operations in the TSP-D formulation are directed, but we will disregard the directions for finding combs. After all, a violated comb inequality in an undirected graph is violated, it is also violated in the directed variant. To find an alternative comb formulation, we look at the tour that is formed by all combined nodes and the operations connecting them. In the terminology of Balas (1995), the truck and drone nodes can be seen as skipped nodes. The following notation is introduced for the TSP-D. $a(S)$ sums the weights of all operations that start and end in subset S. $a(S,T)$ sums the weights of all operations that start in S and end in T, where operations that start and

end in the same node are neglected.

$$a(S) = \sum_{i,j \in S, i \neq j} \sum_{o \in O^-(i) \cap O^+(j)} x_o \qquad \text{for } S \subseteq V \tag{29}$$

$$a(S,T) = \sum_{i \in S} \sum_{j \in T \setminus \{i\}} \sum_{o \in O^-(i) \cap O^+(j)} x_o \qquad \text{for } S,T \subseteq V \tag{30}$$

To derive an alternative formulation for comb inequalities, we also need to make extra assumptions. Whereas using the same edge multiple times might be optimal in the TSP-D, as proven by Agatz et al. (2018), we now assume that an edge can be used only once. Moreover, it is assumed that every node has exactly one ingoing edge and one outgoing edge, i.e. the node is incident to exactly two other nodes if it is visited by the tour. In the original TSP-D, a possible solution is to leave the truck at the depot and visit all customers with the drone, or to use an edge multiple times. In order to use comb constraints, it is assumed that these types of solutions is not possible in this thesis. Equation 31 replaces Equation 14 in the TSP-D formulation. Equation 32 is implied by this equation, the sub-tour constraints and the bound constraints.

$$\sum_{o \in O^+(i) \cap O^-(i)} x_o = 2 \cdot y_i \qquad \forall i \in V \tag{31}$$

$$\sum_{o \in O(ij) \cap O(ji)} x_o \leq 1 \qquad \forall i \in V, j \in V \setminus \{i\} \tag{32}$$

To derive comb constraints for the TSP-D, we sum Equation 31 for all nodes in the handle and sum Equation 32 for all edges that form the teeth. This yields the following inequality:

$$\sum_{i \in H} \sum_{o \in O^+(i) \cap O^-(i)} x_o + \sum_{k=1}^{s} \sum_{(i,j) \in T_k} \sum_{o \in O(ij) \cap O(ji)} x_o \leq 2 \cdot \sum_{i \in H} y_i + s$$

Both sides of this constraint are divided by 2 and rewritten in the symbols used by Balas (1995). As the first term sums all weights starting and ending in H, it is the same as twice the sum of the weights of all edges within nodes in H added to the sum of the weights of all edges having a start or end node in H, yielding $2 \cdot a(H) + a(H, V \setminus H)$.

$$\frac{1}{2} \cdot \sum_{i \in H} \sum_{o \in O^+(i) \cap O^-(i)} x_o + \frac{1}{2} \cdot \sum_{k=1}^{s} \sum_{(i,j) \in T_k} \sum_{o \in O(ij) \cap O(ji)} x_o \leq \sum_{i \in H} y_i + \frac{1}{2} \cdot s$$

$$\frac{1}{2} \cdot \Big(2 \cdot a(H) + a(H, V \setminus H)\Big) + \frac{1}{2} \cdot \sum_{k=1}^{s} a(T_k) \leq \sum_{i \in H} y_i + \frac{1}{2} \cdot s$$

$$a(H) + \frac{1}{2}\Big(\sum_{k=1}^{s} a(T_k) + a(H, V \setminus H) - \sum_{k=1}^{s} a(T_k)\Big) + \frac{1}{2} \cdot \sum_{k=1}^{s} a(T_k) \qquad \leq \sum_{i \in H} y_i + \frac{1}{2} \cdot s$$

$$a(H) + \sum_{k=1}^{s} a(T_k) + \frac{1}{2} \cdot \Big(a(H, V \setminus H) - \sum_{k=1}^{s} a(T_k)\Big) \qquad \leq \sum_{i \in H} y_i + \frac{1}{2} \cdot s$$

It is important to note that $a(H, V \setminus H) - \sum_{k=1}^{s} a(T_k)$ is always positive. As the teeth are a subset of all edges going out of H, $\sum_{k=1}^{s} a(T_k)$ is always smaller than $a(H, V \setminus H)$. Therefore, we can safely round the coefficient $\frac{1}{2}$ down to 0. As s is odd by definition, rounding down $\frac{1}{2} \cdot s$ yields $\frac{s-1}{2}$. Therefore, rounding all coefficients down to the nearest integer yields the blossom constraint in Equation 33.This equation is similar to the blossom constraints used by Gendreau et al. (1997), Fischetti et al. (1997), Fischetti et al. (1998), and Bérubé et al. (2009). The final blossom inequality using the notation of Agatz et al. (2018) yields Equation 34.

$$a(H) + \sum_{k=1}^{s} a(T_k) \qquad \leq \sum_{i \in H} y_i + \frac{s-1}{2} \tag{33}$$

$$\sum_{i,j \in H} \sum_{o \in O^-(i) \cap O^+(j)} x_o + \sum_{k=1}^{s} \sum_{i,j \in T_k} \sum_{o \in O^-(i) \cap O^+(j)} x_o \qquad \leq \sum_{i \in H} y_i + \frac{s-1}{2} \tag{34}$$

This inequality applies to blossoms only, but can be generalized for (simple) combs as Equation 35 (Bérubé et al., 2009).

$$\sum_{i,j \in H} \sum_{o \in O^-(i) \cap O^+(j)} x_o + \sum_{k=1}^{s} \sum_{i,j \in T_k} \sum_{o \in O^-(i) \cap O^+(j)} x_o \leq \sum_{i \in H} y_i + \sum_{k=1}^{s} |T_k| + \frac{s+1}{2} \tag{35}$$

4.3 Logical inequalities

Next to comb inequalities, branch-and-cut algorithms from the literature also implement other cuts. The cut family of logical inequalities was introduced for the OP by Leifer and Rosenwein (1994) and can be directly transferred to the TSP-D. As in the OP, we introduce $\delta(i)$ to denote all edges going out of node i. The logical inequality in Equation 36 is used for the OP and PCTSP.

$$x_{ij} \leq y_j \qquad \forall (i,j) \in \delta(j), j \in V \setminus \{1\} \tag{36}$$

In essence, it defines explicitly what is implied by the cover, count, sub-tour, and bound constraints in the integer problem formulation. However, as the sub-tour and bound constraints

do not necessarily hold in the LP relaxation of the subproblems, logical inequalities can be violated. This reasoning also applies to the TSP-D, as it was assumed earlier that there can be at most one operation going from customer node i to j, or the other way around, and as both i and j have to be combined nodes if the tour uses an operation between these nodes. This yields the logical inequality in Equation 37.

$$\sum_{o \in O(ij) \cup O(ji)} x_o \leq y_i \qquad \forall (i,j) \in \delta(i), i \in V \setminus \{v_0\} \tag{37}$$

Gendreau et al. (1998), Fischetti et al. (1998), and Bérubé et al. (2009) all separate logical inequalities by complete enumeration. This method is also applied in this thesis and will not be discussed in the next section on cut separation because of its simplicity.

5 Separation algorithms

In this section, we introduce a separation algorithm for sub tour constraints and multiple ways to identify comb inequalities.

5.1 Separation of sub-tour inequalities

The following sub-tour constraints were derived for the TSP-D by Agatz et al. (2018):

$$\sum_{o \in O^+(S)} x_o \geq y_i \qquad \forall S \subset V \setminus \{v_0\}, i \in S \tag{38}$$

An exact method for finding the sub-tour constraints to be added to the LP was developed by P.C. Bouman (personal communication, April 16, 2018). This method proposes a Mixed Integer Program (MIP) and returns a subset S that violates the sub-tour inequality if the objective is negative. As this method is exact and generally well performing, it is incorporated in the branch-and-cut algorithm in this research. First, the coefficients c_{ij} are defined as:

$$c_{ij} = \sum_{o \in O^-(i) \cap O^+(j)} x_o \tag{39}$$

The decision variables z_{ij} and w_i are introduced to find the set S and node i for which the sub-tour inequality is most violated. Variable z_{ij} is defined as $z_{ij} := \max\{0, w_j - w_i\}$. From

the MIP in Equations 40-43, the resulting subset includes all nodes i with $w_i = 1$. For every location in the subset, the violation of the sub-tour inequality is assessed. If it is positive, the corresponding sub-tour inequality is added to the formulation, otherwise, it is added to the cut pool.

$$\min_{z,w} \quad \sum_{(i,j)\in V'} c_{ij} z_{ij} - \max_{i \in V'} y_i w_i \tag{40}$$

$$\text{s.t.} \quad z_{ij} \geq w_j - w_i \qquad \forall i, j \in V', i \neq j \tag{41}$$

$$z_{ij} \in \{0, 1\} \qquad \forall i, j \in V \tag{42}$$

$$w_i \in \{0, 1\} \qquad \forall i \in V' \tag{43}$$

5.2 Separation of blossom and simple comb inequalities

This thesis assesses two methods to find violated comb inequalities. The first method is based on the separation algorithm for finding a violated sub-tour inequality in Section 5.1, as it solves a MIP to find the most severe violation. The second method is the heuristic comb-finding algorithm by Padberg and Rinaldi (1990b), which was also used by Bérubé et al. (2009). It is expected that the heuristic will find violated cuts faster, but that the cuts found by the MIP are generally stronger.

5.2.1 Exact separation of blossom inequalities

To find comb inequalities, a similar method is used. A violated comb is found when the objective value of the Mixed Integer Program below is positive. Let w_i and u_{ij} be decision variables indicating whether node i is in the handle and indicating whether node i and j form a tooth, respectively. In u_{ij}, i is the node contained in the handle and j is the node outside of it. Variable θ_{ij} is used to indicate whether both node i and j are included in the handle. Furthermore, parameter f_{ij} is used as expressed in Equation 44. Constraints 47-50 define the variables used in the MIP. If the optimal solution value is found to be positive, a violated blossom inequality has been identified. The handle H and set of teeth $\{T_1, \ldots, T_s\}$ can then be found as $H = \{i : i \in V, w_i = 1\}$ and $\{T_1, \ldots, T_s\} = \{(i, j) : (i, j) \in V, u_{ij} = 1\}$. As the teeth are disjoint sets of two nodes, it is straightforward to identify them individually.

It is emphasized that the MIP in Equations 45-53 will only identify blossom inequalities with three teeth. As there is no linear way to check whether $\sum_{i \in V} \sum_{j \in V \setminus \{i\}} u_{ij}$ is odd, this is not incorporated in this MIP. We present ways to deal with this restriction later in this thesis.

$$f_{ij} = \sum_{o \in O^-(i) \cap O^+(j)} x_o \tag{44}$$

$$\max \quad \sum_{i \in V} \sum_{j \in V \setminus \{i\}} \theta_{ij} \cdot f_{ij} + \sum_{i \in V} \sum_{j \in V \setminus \{i\}} u_{ij} \cdot (f_{ij} + f_{ji}) - \sum_{i \in V} w_i \cdot y_i - \frac{1}{2}\Big(\Big(\sum_{i \in V} \sum_{j \in V \setminus \{i\}} u_{ij}\Big) - 1\Big) \tag{45}$$

$$\text{s.t.} \quad \sum_{i \in V} \sum_{j \in V \setminus \{i\}} u_{ij} \geq 3 \tag{46}$$

$$\sum_{j \in V \setminus \{i\}} \theta_{ij} \leq |V| \cdot w_i \qquad \forall i \in V \tag{47}$$

$$\sum_{i \in V \setminus \{j\}} \theta_{ij} \leq |V| \cdot w_j \qquad \forall j \in V \tag{48}$$

$$\sum_{j \in V \setminus \{i\}} u_{ij} \leq w_i \qquad \forall i \in V \tag{49}$$

$$\sum_{i \in V \setminus \{j\}} u_{ij} \leq 1 - w_j \qquad \forall j \in V \tag{50}$$

$$w_i \in \{0, 1\} \qquad \forall i \in V \tag{51}$$

$$u_{ij} \in \{0, 1\} \qquad \forall i, j \in V, i \neq j \tag{52}$$

$$\theta_{ij} \in \{0, 1\} \qquad \forall i, j \in V, i \neq j \tag{53}$$

5.2.2 Separation heuristic for blossom and simple comb inequalities

As solving the MIP formulation in the previous section takes much computational time, we also consider a heuristic to find comb constraints. Next to the blossom constraints, this algorithm also considers simple comb constraints. As in Bérubé et al. (2009), Gendreau et al. (1997) and Gendreau et al. (1998), simple comb constraints are separated by the algorithm from Padberg and Rinaldi (1990b) in this thesis. The input of the algorithm is the instance graph G, the current solution (x, y) and the fractional graph F that is induced

by the edges with a fractional weight in x. The list of possible handles ($\mathcal{H}$) is based on the block decomposition of F, which is the division of the graph in biconnected components. A biconnected component of a graph is a subgraph that is maximally biconnected, meaning that it is still connected when one edge is removed. All blocks containing more than 3 nodes are put in the set $\mathcal{B}$. Possible handles (H) are constructed by taking the power set of $\mathcal{B}$ and therefore, the set of possible handles contains all combinations of blocks with more than 3 nodes. $\mathcal{H}$ is then defined as the set of all H. If the induced subgraph of a particular $H \in \mathcal{H}$ is connected, we further investigate the possibility of making a comb with H as handle. For a certain handle H, a set of virtual edges $\mathcal{T}_H$ is constructed. All edges that are incident to H and have a weight of 1 in x are added to this set. Padberg and Rinaldi (1990b) then add a virtual edge to $\mathcal{T}_H$ for every cut-node in H with an even degree in F. A cut node is a node that connects two biconnected components. To find a virtual edge for a cut node v in H, a set of possible virtual edges J_v is constructed. This set consists of all regular edges incident to v and another end outside of H, biconnected components of F intersecting H in v, and intersections of a block in F with the neighborhood set of node v. The last two possibilities enable this algorithm to find simple combs. Among the possible virtual edges $\mathcal{J}_v$, the virtual edge with maximum weight is added to $\mathcal{T}_H$, where the weight is defined as in Equation 54.

$$x_T = a(T, T) + 2 - |T| \tag{54}$$

After the set of virtual edges is filled, the elements with $x_T \geq 0.5$ are added to the set of teeth $\mathcal{T}$. If $|\mathcal{T}|$ is even, we go back to the set of virtual edges $\mathcal{T}_H$ and add the virtual edge T' such that $\min\{x_{T'}, 1 - x_{T'}\} \geq \min\{x_T, 1 - x_T\} \forall T \in \mathcal{T}_H$, i.e. such that its weight is the most fractional. Depending on whether this virtual edge was already in $\mathcal{T}$, we add it to or remove it from $\mathcal{T}$. The possible comb now has an odd number of teeth, but the teeth might be overlapping. Overlapping teeth are removed from the set of teeth and included in the handle. To increase the possibility of finding a comb with this algorithm, an extra step is added to Algorithm 1. After detecting the overlapping teeth, the remaining teeth are collected. If the number of remaining teeth is even, we can obtain an odd number of teeth by adding one of the overlapping teeth. Finally, we check if the comb is violated by computing the both sides of the inequality in Equation 35. For all used definitions, assumptions and

other implementation details the reader is referred to Padberg and Rinaldi (1990b). The algorithm presented here is used as a method to find comb inequalities fast, however, it is not exact. Therefore we expect it to find less combs than the exact separation procedure presented in the previous section. However, the proposed algorithm is much faster than the exact separation procedure. Algorithm 1 summarizes the used heuristic.

Algorithm 1: Algorithm for simple comb separation, as in Padberg and Rinaldi (1990b)

Input: Graphs G and F and solution (x, y);
Define the set blocks of F with more than 3 nodes $\mathcal{B}$;
Construct the list of possible handles $\mathcal{H}$ by taking the power set of $\mathcal{B}$;
for $H \in \mathcal{H}$ **do**
    Consider component $H \in \mathcal{H}$;
    if H *is connected* **then**
        Add all 1-edges incident to H to set $\mathcal{T}_H$;
        for $v \in \mathcal{H}$ *such that* $|\delta(v)|$ *is even* **do**
            Construct set of possible virtual edges $\mathcal{J}_v$ as explained above ;
            Add the possible virtual edge with maximum weight x_T to $\mathcal{T}_H$;
        Add all elements in $\mathcal{T}_H$ with $x_T \geq 0.5$ to the set $\mathcal{T}$;
        if $|\mathcal{T}|$ *is even* **then**
            Determine T' as the most fractional virtual edge ;
            if $T' \in \mathcal{T}$ **then**
                Set $\mathcal{T} = \mathcal{T} \setminus T'$;
            else
                Set $\mathcal{T} = \mathcal{T} \cup \{T'\}$;
        while *a set of teeth* $\{T_i, \ldots, T_j\}$ *intersects* **do**
            Set $\mathcal{T} = \mathcal{T} \setminus \{T_i, \ldots, T_j\}$ and $H = H \cup \{T_i, \ldots, T_j\}$;
        if *comb inequality is violated by current solution and* $|\mathcal{T}| \geq 3$ **then**
            Record the comb inequality ;
        else
            continue;

5.2.3 Using heuristics to create variants of combs

As solving a MIP to find violated combs takes quite some effort, this thesis proposes using heuristics to find combs that are similar to the comb found by the MIP proposed in Section 5.2.1. The combs found by these heuristics are only added if they are violated in the current node of the branch-and-cut tree. As the MIP only finds the most violated comb, there might be other violated comb inequalities that are also violated by the current LP solution or later in the algorithm. To this end, we use the idea of a cut pool that was introduced by Padberg and Rinaldi (1991). We propose to create variants of a found comb and put these variants in a cut pool. In every following search for violated cuts, the cut pool will first be searched for violated inequalities. This is a fast and efficient way to find combs. In this section, we will introduce the heuristics used to create the variants. As a reference, Figure 9 shows an example of a comb. The first heuristic approach considered in this thesis is deleting nodes that are only contained in the handle from the comb. This is depicted in Figure 10. This approach yields $2^{|H|-s} - 1$ variants of the base comb, as it includes all subsets of the set $H \setminus \bigcup_{j=1}^{s} T_j$ except the subset containing all nodes of the considered set. The second heuristic approach is depicted in Figure 11 and includes nodes outside the comb in the handle. This approach yields $2^{|V|-|H|-\sum_{j=1}^{s}|T_j|+s} - 1$ variants of the base comb. As both of these heuristic approaches can result in many variants of the same comb, we implement them only if $|H| - s \leq 10$ and $|V| - |H| - \sum_{j=1}^{s} |T_j| + s \leq 10$, respectively. Therefore, each of the heuristics will not return more than 1024 constraints in an iteration. Variants of a base blossom can also be obtained by swapping nodes within a tooth. The tooth node that was initially contained in the handle is now excluded from the handle and vice versa , as depicted in Figure 12. This approach yields $2^s - 1$ variants of the base blossom. Tooth nodes that are not included in the handle can also be exchanged in between tooths, as depicted in Figure 13. This yields $s! - 1$ extra variants. The same nodes can also be replaced by nodes outside of the comb, as depicted in Figure 14. The number of variants created can then be computed by

$$\sum_{j=1}^{\min\{|V|-|H|-s,s\}} \binom{|V| - |H| - s}{j} \cdot \binom{s}{j}$$

For the simple combs from the Padberg and Rinaldi (1990b) algorithm, only variants are created following Figures 10, 11, and 13 if the comb has three teeth.

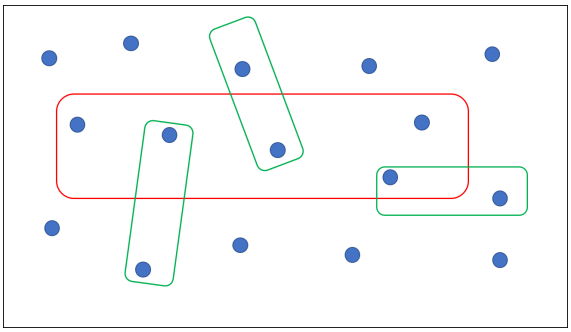


Figure 9: Example of a comb in an arbitrary instance

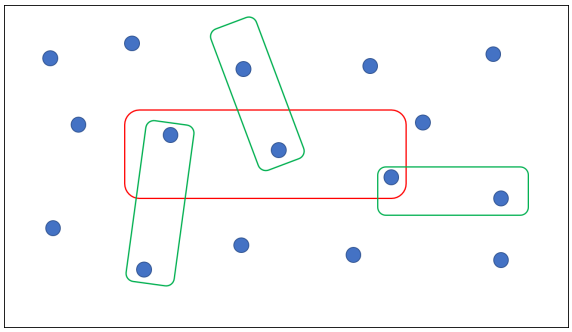


Figure 10: Creating a variant by deleting nodes from handle

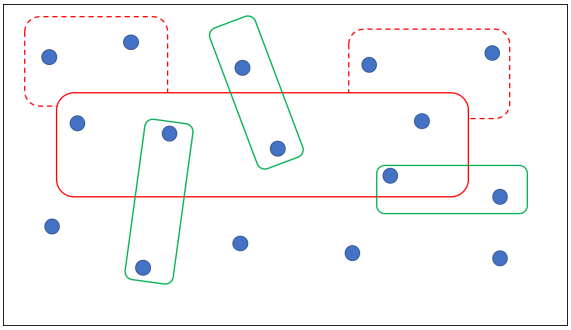


Figure 11: Creating a variant by including loose nodes

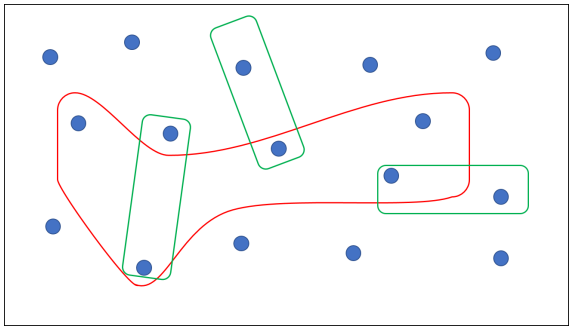


Figure 12: Creating a variant by swapping tooth nodes

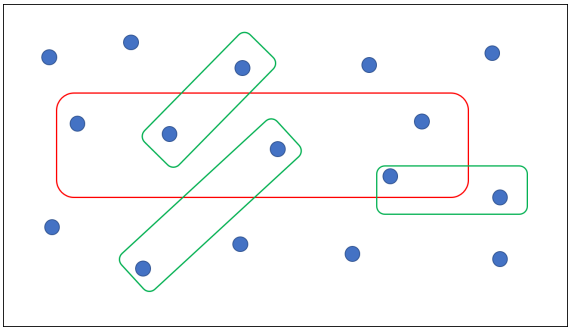


Figure 13: Creating a variant by swapping teeth

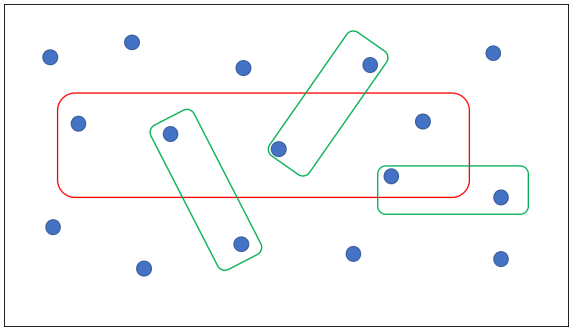


Figure 14: Creating a variant by changing teeth

5.2.4 Obtaining combs with an odd number of teeth

As it is not possible to capture the criterion for an odd number of teeth in a linear constraint, the MIP can also find combs with an even number of teeth. As these do not yield valid inequalities, they are not useful in solving the TPS-D. In this thesis, we consider three methods to overcome this problem. In the first method the MIP from Section 5.2.1 is solved to optimality. When the resulting blossom has an even amount of teeth, as in Figure 15, valid blossoms can be obtained by creating an extra tooth, as in Figure 16, or deleting a tooth, yielding the comb in Figure 9. These variants are saved in the cut pool, as they have a higher probability of being violated later on. The advantage of this method is that cuts can be found in a later stadium without having to solve a MIP, which is quite time consuming. However, this method might miss combs with an odd number of teeth that violate the inequality slightly less than a comb with an even number of teeth.

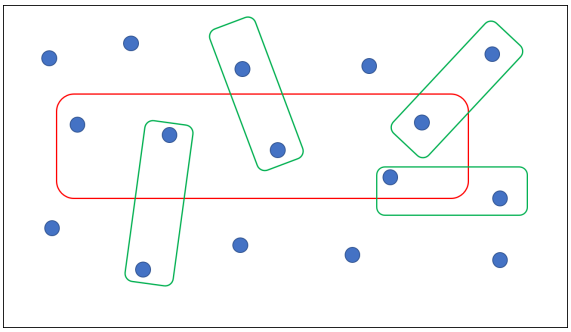


Figure 15: Example of a comb with even number of teeth

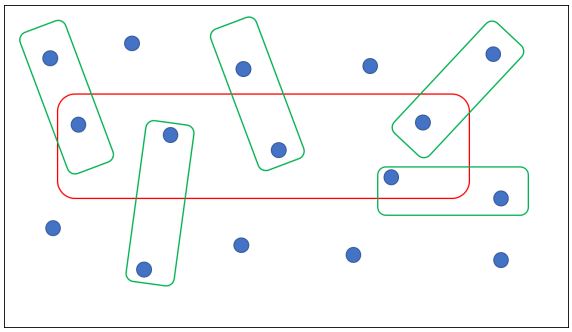


Figure 16: Creating a variant by adding an extra tooth

The second method implicitly adds the non-linear constraint to the MIP using an `Incumbent Callback` in CPLEX. This callback function rejects all encountered incumbents that have an even number of teeth. This callback function can be used while solving the MIP from Section 5.2.1 to optimality. This method finds all combs with a positive violation and an odd number of teeth and is therefore exact. However, it does not use information from combs with an even number of teeth, as the first method does. The third method resembles the second method as the same `Incumbent Callback` is used to reject combs with an even number of teeth. However, instead of solving the separation MIP from Section 5.2.1 to optimality, the

CPLEX function `populate` is used to save all incumbents encountered when solving the MIP. Due to the θ variables in the MIP formulation, there are many incumbent solutions yielding the same comb. Therefore, only unique combs are saved and checked for a positive violation. This method is exact and will not miss any comb, but it takes more time as all combs are found by forming a solution pool for a MIP. As the last two methods introduce an `Incumbent Callback`, some other solving features of CPLEX are disabled. To limit the computational time spent in each node, a 5 minute time limit is set to solve the MIP when an `Incumbent Callback` is used.

6 Implementation of the branch-and-cut algorithm

This section describes the branch-and-cut algorithm in detail and elaborates on all implementation details. This approach is based on the algorithms used by Fischetti et al. (1997), Fischetti et al. (1998) and Bérubé et al. (2009). Figure 1 in Section 3.1.1 shows an outline of the algorithm. The branch-and-cut algorithm is implemented by CPLEX version 12.8 and was written in Java 8.

6.1 Branch-and-cut algorithm in CPLEX

A MIP corresponding to the formulation in Section 3.4 was constructed to model the TSP-D. We solve this MIP using a branch-and-cut solver in CPLEX. This section elaborates on the working of this solver in CPLEX. Initially, only the sub-tour constraints corresponding to a set of only one location were added to the MIP. Other sub-tour inequalities are added later in the algorithm by the separation algorithm described in 5.1. After the initialization phase there is only one node in the branching tree. This node is then automatically selected in this step of the algorithm. However, after some iterations, there are multiple nodes that can be investigated. The node with the lowest bound is selected for branching, as is the default option in CPLEX. If the list of nodes is empty, the current best upper bound is the optimal solution. At this stage, the algorithm searches for violated inequalities, which we will discuss in detail in the next section. Next, the LP of the current node is solved with CPLEX. The upper and lower bounds of the nodes preceding the current node in the branching tree are

updated with the newly found bounds. If the lower bound of this node is higher than the best known upper bound, there is no reason to investigate this node further, and the node is pruned. If it is lower than the best known upper bound, there is still room for improvement and the algorithm continues to eventually branch on this node. When a node is pruned, this node is not investigated more. After pruning, another node in the list is selected to be solved. If this list is empty, the problem is solved to optimality. In CPLEX, priority is given to branching on the y_i variables, as they impose restrictions on all x_o starting and ending in customer i as well, as in Bérubé et al. (2009).

6.2 Finding cuts

Sub-tour inequalities can be added using both `Lazy Constraint Callbacks` and `User Cut Callbacks`. Whereas the `Lazy Constraint Callbacks` cut the integer solution space and are only called when the algorithm has found an integer solution, the `User Cut Callbacks` only cut off fractional solutions and are called at any point in the algorithm. As the sub-tour inequalities are actually a part of the TSP-D formulation in Section 3.4 and we would like the algorithm to find violated sub-tour inequalities in every node, we have chosen to implement the callback for these cuts as both `Lazy Constraint Callbacks` and `User Cut Callbacks`. The comb inequalities and logical inequalities are added as `User Cut Callbacks` as they only tighten the integer solution space. When using callbacks, one can define how CPLEX should treat the found inequalities. The setting `Use Cut Force` forces CPLEX to add the found inequality to the MIP formulation and keeping it there. When using the setting `Use Cut Purge`, CPLEX can choose to remove it from the MIP formulation if it is redundant. Another important setting in cut separation is the order in which the different separation algorithms are called, as the algorithms are assessed sequentially and the finding of cuts is ceased whenever a violated cut is found. For this thesis, the order is set to:

1. Existing constraints in the cut pool
2. Separation of logical constraints
3. Separation of sub-tour constraints
4. Heuristic separation of comb inequalities

5. MIP separation of comb inequalities

This order is based on the strength of the cuts and the speed of the separation algorithms. The logical constraints are relatively easily found by an exact algorithm and can affect the fractional solution significantly. The sub-tour constraints are important as they are a part of the MIP formulation and therefore might cut off integer solutions. If no cuts are found up until this point, the separation algorithm of Padberg and Rinaldi (1990b) is executed. As this is an heuristic approach, finding a violated comb is not guaranteed, even if one is present. If the heuristic does not find a simple comb or blossom, the separation method using a MIP for finding blossoms is used as a last resort. Even though this method is more flexible in finding combs than the Padberg and Rinaldi (1990b) algorithm, one cannot guarantee that a violated comb is always found and solving the MIP takes quite some computational time. The cut pool that is assessed first in this phase is filled with variants of the constraints found by the other separation algorithms. For comb constraints, the variants from Section 5.2.3 are added to the pool. For sub-tour inequalities, the variants consists of sub-tour inequalities with a combination of set and location that was not violated at the time of creation.

6.3 Experiment design

This research is aimed at implementing a branch-and-cut algorithm for the TSP-D. To this end, the quantitative analysis is split in two. From the preliminary analysis we would like to find out if the branch-and-cut algorithm works for the TSP-D and assess the effect of the proposed inequalities. In order to find the best configuration, we compare the averages of the number of nodes in the branch-and-cut tree and the number of user cuts in several configurations. Version A uses only the sub-tour inequalities, whereas version B also uses logical inequalities. The other versions all implement sub-tour, logical and comb inequalities. Version C implements corresponds to not including an `Incumbent Callback` but using information from combs with an even number of teeth to fill the cut pool. Version D corresponds to solving the MIP with the `Incumbent Callback` and version E uses the MIP and the `Incumbent Callback` to fill a solution pool. Version F only uses the Padberg and Rinaldi (1990b) algorithm to find violated comb inequalities. Table 3 shows the different variants

tested in this analysis.

Table 3: Tested versions

Version	`Lazy cuts`	`User cuts`	Comb separation method
A	Sub-tour cuts	Sub-tour cuts	-
B	Sub-tour cuts	Sub-tour, logical cuts	-
C	Sub-tour cuts	Sub-tour, logical, comb cuts	Padberg and Rinaldi (1990b), MIP separation
D	Sub-tour cuts	Sub-tour, logical, comb cuts	Padberg and Rinaldi (1990b), MIP separation with `Incumbent Callback`
E	Sub-tour cuts	Sub-tour, logical, comb cuts	Padberg and Rinaldi (1990b), MIP separation with solution pool and `Incumbent Callback`
F	Sub-tour cuts	Sub-tour, logical, comb cuts	Padberg and Rinaldi (1990b)

The versions are tested for problem instances with increasing number of customer nodes and increasing difficulty of the operations that are allowed. From a problem instance with 15 customer nodes and no operations with truck only nodes allowed, we let the number of customer nodes (n) increase as $\{15, 20, 25, 30, 40, 50\}$ and the number of truck nodes in an operation (t) up to 5. It is important to note that customer nodes are added to existing instances, i.e., the first generated instance with 20 customer nodes contains all customer nodes from the first instance with 15 customer nodes and 5 additional customers. 30 Instances were run and compared for every configuration on the number of nodes in the branch-and-cut tree (N), the number of user cuts added (U) and the gap between the optimal solution value and the lower bound in the root node (G). Statistical tests are performed to assess the distribution and significance of the differences in the number of nodes of the branch-and-cut tree. These differences are tested for normality with the Shapiro-Wilk test. The null hypothesis (H_0) in this test is that the sample data is normally distributed. As the popular t-test assumes a normal distribution of the sample and the data, as is shown later, does not generally follow a normal distribution, the Wilcoxon signed rank test is used to test whether the sample differences are symmetrically distributed around zero (H_0). The

Wilcoxon signed rank test approximates the normal distribution when there are more than 20 nonzero differences recorded. In that case, p-values are calculated. Otherwise, the test statistic is compared to a more sophisticated distribution. Whenever the Shapiro-Wilk test shows that a normal distribution cannot be rejected, the t-test is used to test whether there is a significant difference between the two versions. Finally, we will test the sensitivity of the algorithm to changes in the number of decimals the costs are specified in and to enabling `Use Cut Purge`. This setting allows CPLEX to remove the constraint if it is ineffective in a later stadium. In the extensive results, we investigate a way to maximize the benefit by running two configurations in parallel in an optimistic environment where violated cuts can be identified immediately. As parallelization is an important trend in hardware development, we hope to find that it can also improve the search to an optimal solution for the TSP-D. As the separation of more types of inequalities generally leads to a higher computational time, we will exclude the time spent in separation in this analysis. In practice, this means that two variants solve a particular instance in parallel. Whenever one of the variants reached the solution of an instance, the time spent excluding callbacks is stored, and the computation of the other version is terminated upon reaching this time limit. Assessing the optimality gap at this point gives an indication of how close the other version is to completely solving the problem.

6.4 Instance generation

To test the effectiveness of the proposed cut families in a branch-and-cut algorithm, we use generated instances of the TSP-D of different sizes and difficulty levels. For the generation of an instance, locations are placed uniformly in the [0,1]x[0,1] plane. The first location is automatically marked as the depot. The locations are then used to define the operations as explained in Section 3.4. We run instances with operations that have up to 5 truck only nodes allowed. Operations with more than 5 truck only nodes are not considered in this thesis as allowing an infinite amount of truck nodes increases the solution time but does not yield a better solution in general. (Bouman et al., 2017) As Agatz et al. (2018), this thesis aims at minimizing the total travel time. Initially, it is assumed that the travel time of the truck only depends on the Euclidean distance between two locations and that the drone is

twice as fast as the truck.

7 Computational results

This section is concerned with presenting the results from the performed analyses. All experiments were performed on the Lisa computer cluster of SURFsara. Instances were solved on either 32GB or 64GB nodes and solving instances was done in parallel if possible. The computer cluster runs on Linux, which has implications for the working of CPLEX 12.8. When running on other operational systems, differences were observed regarding the moments `User Cut Callbacks` were called, choosing the optimal solution for the separation MIP when multiple solutions yield the same objective value and branching strategies. The optimal objective value and solution of the TSP-D remained constant over different operating systems. This section is split into preliminary results, concerned with an overall analysis of the configurations in which the proposed valid inequalities work best, and extensive results, where the best performing configurations are inspected in detail.

7.1 Preliminary results

In this section, the results obtained from the experiments are discussed. First, we inspect the behavior of the different versions in case we increase the number of customer nodes n and the number of allowed truck nodes t. Next, we assess the effect of letting CPLEX delete redundant constraints during solving. Finally, the instances with 30 to 40 customer nodes are inspected in detail. Based on these analyses, we will choose the best performing configurations for the extensive results. All numeric results reflected in the tables and graphs in this section can be found in Appendix B.1.

7.1.1 Comparing the branch-and-cut algorithm to existing literature

First and foremost, we observe that the branch-and-cut algorithm is very suitable for solving the TSP-D. Tables 4 and 5 show the solution times for several versions of the branch-and-cut algorithm. Versions D and E are not shown in these tables as version C proved to be the fastest algorithm yielding a similar size of the branch-and-cut tree. Version F is not shown

as it generally did not find comb inequalities. An asterisk is reported when the available memory was insufficient.

Table 4: Solution times (minutes:seconds) using `Use Cut Force`, averages of 30 instances

	$t = 0$					$t = 1$			
n	15	20	25	30	40	15	20	25	30
A	00:01	00:03	00:10	00:21	01:49	00:07	00:36	02:13	09:47
B	00:01	00:03	00:08	00:21	01:58	00:07	00:37	02:26	07:52
C	00:05	00:19	00:57	02:17	*	00:21	01:44	10:38	*

Table 5: Solution times (minutes:seconds) using `Use Cut Force`, averages of 30 instances

	$n = 20$				$n = 25$	
t	1	2	3	4	1	2
A	00:36	02:33	10:34	20:22	02:13	18:17
B	00:37	02:47	10:12	22:26	02:26	19:58
C	01:44	05:24	14:32	30:17	10:38	*

Up until this point, the best performing exact algorithm for the TSP-D was the A* algorithm used by Bouman et al. (2017). That algorithm solved instances with 20 customer nodes and 2 truck nodes, on average, in 10:35:26. The results above show that the branch-and-cut algorithm is very time-efficient. For the comb inequalities, instances with many customer nodes and/or multiple allowed truck nodes caused memory problems, retaining the algorithm from solving in versions C, D, E and F. Nevertheless, the results prove that the branch-and-cut algorithm is an efficient way of solving TSP-D instances to optimality.

7.1.2 Comparison of variants for different instance sizes

In this section, all variants explained in Section 6.3 are compared by using them to solve the same instances. For all configurations considered in this paragraph, the setting `Use Cut Force` is used in CPLEX, i.e. no redundant constraints are deleted during the branch-and-cut algorithm. Table 6 shows the averages for all versions over 30 instances.

Table 6: Number of nodes in the branch-and-cut tree N and number of user cuts added U, using $t = 0$ and `Use Cut Force`, averages of 30 instances

n	15		20		25		30		40	
Version	N	U	N	U	N	U	N	U	N	U
A	39.20	123.47	70.57	244.20	128.67	401.50	113.20	468.93	369.73	1068.60
B	33.77	128.23	62.13	219.67	107.33	334.40	106.17	412.30	501.17	875.90
C	33.77	128.50	63.30	223.23	102.47	324.63	107.03	428.50	*	*
D	33.70	128.33	62.30	220.47	109.57	326.63	104.40	444.43	*	*
E	33.73	128.43	62.17	220.93	104.37	326.03	104.13	443.87	*	*
F	33.77	128.23	62.13	219.67	107.17	334.23	106.17	412.30	*	*

Using additional types of cuts leads in some cases to a lower average number of nodes in the branch-and-cut trees. This does not imply that these versions outperform variants without these constraints, as there are instances for which the addition of logical and comb inequalities resulted in a larger branch-and-cut tree for every tested n. A more remarkable effect observed in Table 6 is the jump in N from $n = 30$ to $n = 40$, as $N_A > N_B$ for $n \leq 30$, but the reverse is true for $n = 40$. This implies that the logical inequalities only work well for instances with $n \leq 30$. The instances with $n = 30$ and $n = 40$ are investigated in detail later in this chapter, as well as instances with $n \in [32, 34, 36, 38]$. The average number of user cuts needed is generally lower for the versions using logical inequalities. This effect applies through all instance sizes tested. Furthermore, it is observed that, based on the average number of nodes in the branch-and-cut tree and number of user cuts used, there is little difference between versions B, C, D, E and F. In fact, version F differs only from version B for $n = 25$ and $t = 0$. In that configuration, two combs are found by the Padberg and Rinaldi algorithm. This has very limited impact on the measures and therefore, we will not consider version F in the remainder of this thesis.

Moreover, we observe that the average number of nodes in the branch-and-cut tree is similar for all versions when comparing $n = 25$ to $n = 30$. Apparently, the five added customers do not increase the difficulty of the 30 tested instances, when using $t = 0$. From the numeric results in Appendix B.1 it appears that this effect does not hold when allowing more truck

nodes. The most probable cause for this effect is the limited number of instances tested.
As the differences between N for two configurations are both positive and negative, we will test if they follow a normal distribution and if they differ significantly from zero. Ideally, we would like N to decrease when using logical and comb inequalities or when more advanced methods to find combs are used. As such effects cannot be observed directly from the plots, we will use statistical tests to find the general effects. In Table 6, it is observed that averages for $n = 25$ and $n = 30$ show the most differences between versions. Therefore, we will perform the statistical tests on the differences in N between all versions for these instance sizes. Table 7 shows the results of the statistical tests. In this table, the average difference in N is denoted by δ, the p-value for the Shapiro-Wilk test by p_{SW} and the p-value for the Wilcoxon signed rank test by p_W.

Table 7: Average differences in N and p-values

$n = 25$, $t = 0$												
	B			C			D			E		
	$\bar{\delta}$	p_{SW}	p_W	$\bar{\delta}$	p_{SW}	p_W	$\bar{\delta}$	p_{SW}	p_W	$\bar{\delta}$	p_{SW}	p_W
A	21.33	**0.00**	0.09	26.20	**0.00**	**0.04**	19.10	**0.00**	0.18	24.30	**0.00**	0.09
B	-	-	-	4.87	**0.00**	0.47	-2.23	**0.00**	0.10	2.97	**0.00**	0.58
C	-	-	-	-	-	-	-7.10	**0.00**	**0.01**	-1.90	**0.00**	0.92
D	-	-	-	-	-	-	-	-	-	5.20	**0.00**	**sig.**
$n = 30$, $t = 0$												
A	7.03	0.08	0.49	6.17	0.10	0.47	8.8	**0.01**	0.52	9.07	0.06	0.31
B	-	-	-	-0.87	**0.00**	n.s.	1.77	**0.00**	n.s.	2.03	**0.00**	n.s.
C	-	-	-	-	-	-	2.63	**0.00**	n.s.	2.90	**0.00**	n.s.
D	-	-	-	-	-	-	-	-	-	0.27	**0.00**	n.s.

Note: When there are more than 20 nonzero differences recorded a p-value is given for the Wilcoxon signed rank test, otherwise the table shows non-significance (n.s) or significance (sig.). Significance at $\alpha = 0.05$ is marked in bold.

For $n = 25$, Table 6 shows quite some difference between the average number of nodes when using version A compared to the other versions. From the statistical analyses follows that

the differences of none of the combinations are normally distributed for $n = 25$. This is mainly due to the large range in the differences for such a small sample. For example, the differences between version A and B range from -86 to 302. From the Wilcoxon signed rank test followed that the differences between versions A and C, C and D, and D and E do not follow a symmetric distribution around zero. In this respect it can be argued that, on average, version C yields a significantly smaller tree compared to version A and D. Therefore, version C is the most promising configuration, which we will inspect in detail in the extensive results. For $n = 30$, the differences between versions A and B, and versions A and C can be assumed to be normally distributed. The Wilcoxon signed rank test shows that the median of the differences is not significantly different from zero for all combinations of versions. As the results from the statistical results for $n = 30$ differ substantially from those for $n = 25$, it cannot be stated that a configuration outperforms the others for all levels of instance difficulty on the number of nodes in the branch-and-cut tree.

To compare the different configurations for the comb inequalities, we zoom in on the instances with $n = 30$ for which comb inequalities have been used in in Figure 17. Figure 18 shows the average number of comb inequalities that was found for every instance size.

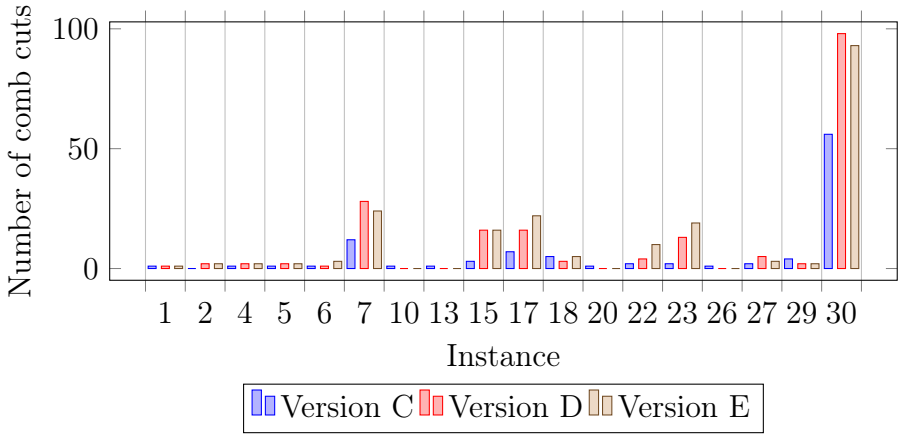


Figure 17: Number of comb constraints for $n = 30$, per instance. Instances where no comb constraints were generated are not shown.

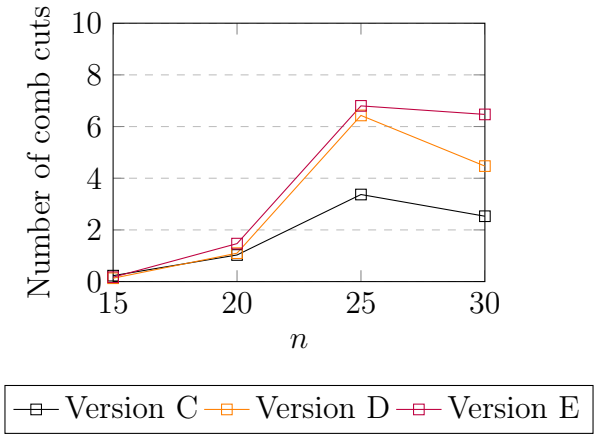
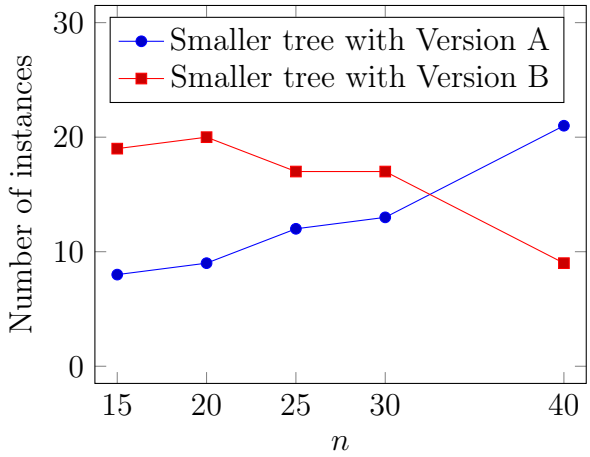


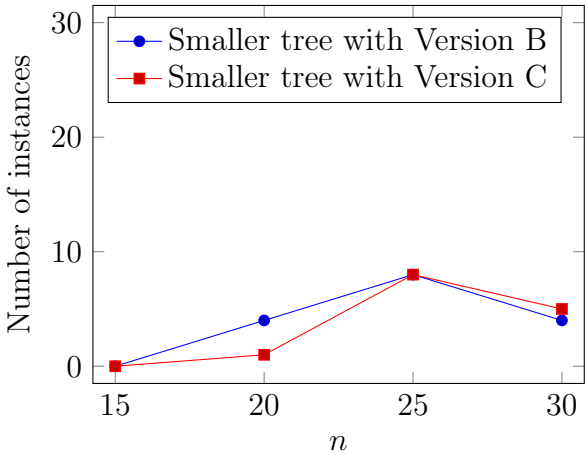
Figure 18: Comparison of number of comb inequalities for different instance sizes, averages over 30 instances

It is observed from Figure 18 that versions D and E generally find the most combs, as was expected. However, it takes much computational time to populate the solution in every callback of a MIP. All versions seem to behave similarly for instances with 15 and 20 customers. Moreover, from Figure 17 it appears that version C sporadically finds more violated cuts by means of the cut pool, caused by the fact that it uses information from invalid combs with an even number of teeth. As the additional cuts found by version D and E do not yield significantly better results in terms of N and the callback functions take much computational time, we shall use version C to find combs in the extensive results.

Finally, we investigate the tested instances more thoroughly for versions A, B and C. The number for which a certain version is smaller is shown in Figure 19.



(a) Comparison of Version A and B



(b) Comparison of Version B and C

Figure 19: The number instances with a smaller tree, $t = 0$. In total 30 instances were tested. Remaining instances had an equal N in both versions.

In Figure 19a we see that most of the times, there are less nodes when using version B. As in Table 6, a remarkable difference is observed between version A and B for $n = 40$. Figure 19b shows more clearly that the comb inequalities only affect the branch-and-cut algorithm for instances with 20 customer nodes or more. For $n = 25$, there are exactly as much trees smaller with version B as there are with version C. For $n = 30$, there are slightly more instances that have a smaller tree in version C.

7.1.3 Effect of allowing truck nodes

In Table 8, numeric results are reported for different numbers of allowed truck nodes (t) when fixing the number of customer nodes at 20.

Table 8: Results using $n = 20$ and `Use Cut Force`, averages of 30 instances

t	0		1		2		3		4	
Version	N	U	N	U	N	U	N	U	N	U
A	70.57	244.20	169.37	294.23	211.40	318.07	212.30	307.43	231.13	314.93
B	62.13	219.67	152.13	282.83	200.63	277.60	234.13	291.43	235.97	296.50
C	63.30	223.23	152.77	282.80	200.20	278.00	234.13	291.73	235.80	296.40
D	62.30	220.47	153.00	283.03	200.63	277.60	234.13	291.73	236.03	296.27
E	62.17	220.93	152.47	283.53	200.63	277.60	234.13	291.60	236.03	296.27

Naturally, the number of nodes in the branch-and-cut tree is larger when allowing one or multiple truck nodes. It seems that the growth in N decreases for a higher t. The number of user cuts does not seem to follow a clear trend. In this analysis, the differences between versions B, C, D and E are even smaller. A logical cause for this result is the used instance size. As combs are based on a separation of the customer nodes, it is straightforward that larger instances yield more possible combs. This also increases the probability of a comb violation. Unfortunately, larger instances could not be solved with the available memory. The largest average difference is observed for $t = 2$ between version A and B, as $\bar{\delta} = 10.77$ for this combination. The Shapiro-Wilk test resulted in a p-value of 0.00, rejecting the null hypothesis of a normal distribution. The p-value of the Wilcoxon signed rank test is 0.052, which is very close to rejecting the null hypothesis of similar medians. Therefore, we will investigate this configuration in the extensive results.

7.1.4 Deleting redundant constraints

For the results in this section, the number of allowed truck nodes is fixed at 0 and the feature `Use Cut Purge` is enabled instead of `Use Cut Force`. This implies that CPLEX can choose to remove added constraints from the MIP formulation. The removed constraints will still be available in the cut pool and are therefore easier to identify if violated in a later stadium, as proposed by Padberg and Rinaldi (1991). Removing constraints yields different behavior of the branch-and-cut algorithm throughout the tree. The experiments from Section 7.1.2 are repeated with this new setting and averages are reported in Table 9.

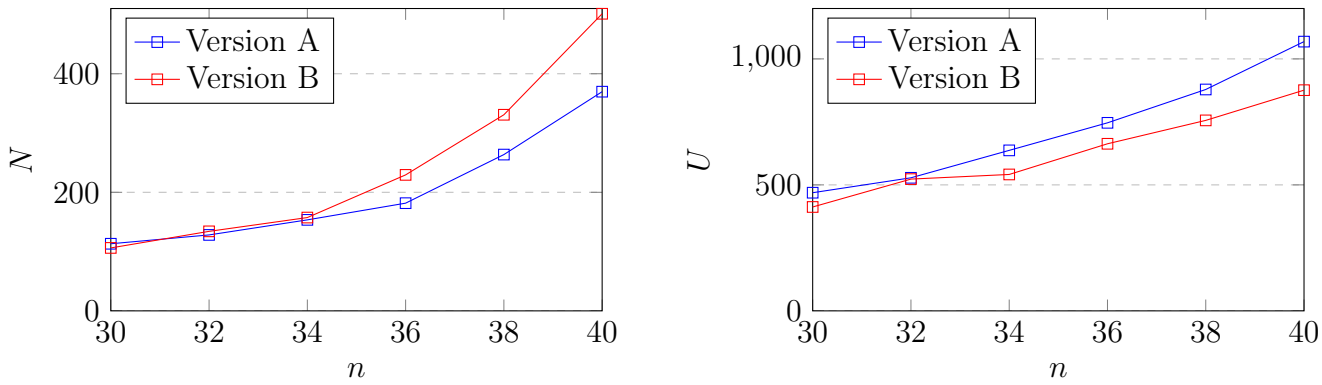
Table 9: Results using `Use Cut Purge`, $t = 0$, averages of 30 instances

n	15		20		25		30		40	
Version	N	U	N	U	N	U	N	U	N	U
A	29.23	149.00	73.50	295.03	125.37	480.90	138.53	649.77	489.47	1272.97
B	34.50	158.93	73.67	263.33	116.33	437.93	125.30	530.30	508.83	1229.07
C	34.53	158.40	72.10	263.83	114.13	442.93	125.07	531.93	*	*
D	34.53	159.07	73.07	265.70	114.83	446.23	128.13	536.03	*	*
E	34.53	159.07	72.90	265.70	116.50	456.63	118.83	541.60	*	*

It is observed that Version B now yields, on average, smaller trees than Version A for $n > 25$ and, comparing the `Use Cut Purge` results to the earlier results, that the average number of nodes increases for all versions for the experiments with $n \geq 20$. Again, these effects do not apply all individual instances. The increase in average number of added cuts was expected, as a particular cut can be added and removed multiple times within a run of the algorithm. As the number of nodes increases for larger instances, we will not perform any extensive analysis with the `Use Cut Purge` setting.

7.1.5 Instances with 30 to 40 customer nodes

In this section, we zoom in on instances with 30 to 40 customer nodes, as a remarkable jump was observed in Table 6. Figure 20 shows the development of N and U for different instance sizes in this interval. Numeric results reflected in this figure are reported in Appendix B.1.



(a) Size of branch-and-cut tree for versions

(b) Average number of cuts used

Figure 20: Averages over 30 instances, $t = 0$

In Figure 20a, N seems to grow exponentially for both variants. Next to that, Version B seems to have a higher growth rate. The two versions yield similar tree sizes for $n = 32$. From these figures, it appears that for instance with more than 32 customer nodes, Version A outperforms Version B. This effect is also seen in Figure 21, depicting the number of trees that have a smaller N in a particular version.

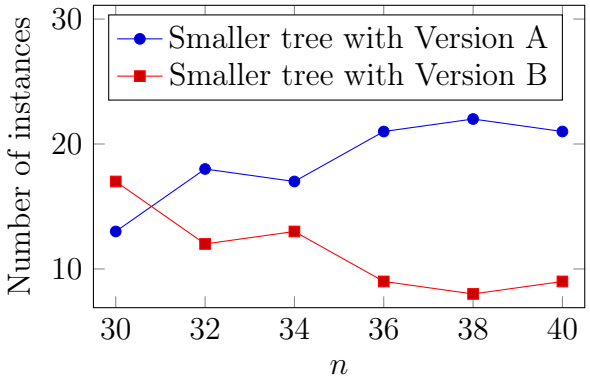


Figure 21: The number instances with a smaller tree, $t = 0$

From $n = 32$ on, there are substantially more instances with a smaller tree when using version A compared to version B. A plausible cause for this effect might be the fact that the number of arcs increases very fast when adding nodes. As the logical constraints are created for single arcs they might not be as powerful as the sub-tour constraints. Moreover, the separation algorithm also prioritizes the separation of logical inequalities, yielding a suboptimal branch-and-cut tree for the majority of the instances. As $n = 30$ seems to be a threshold value, we

will zoom in further on this instance size in the extensive results.

7.1.6 Conclusions from preliminary results

When implementing the order of separation algorithms from Section 6.2, the logical inequalities and sub-tour inequalities prove to be very strong. For small instances, hardly any combs are necessary to find the optimal solution. When a comb was added to the formulation, it had been found most often by the MIP separation from Section 5.2.1, sometimes via the cut pool and sporadically through the Padberg and Rinaldi algorithm. For $t = 0$ and $n = 25$, only 2 comb inequalities were found by this algorithm in Version C in total in 30 instances, comparing to 67 and 32 combs found by the MIP and in the cut pool, respectively. Remarkably, Version D and E did not find any combs by the Padberg and Rinaldi (1990b) algorithm. This might be caused by the fact that they identify violated comb inequalities cutting a similar part of the solution space earlier in the branch-and-cut algorithm, as these versions force any solution to the comb separation MIP to have an odd number of teeth. In general, more combs are found when solving larger instances. However, the number of comb inequalities remains much lower than the number of logical and sub-tour inequalities. Iterating over the cut pool is a fast and effective way to find new cuts of both sub-tour inequalities and comb inequalities. A significant difference was observed between version A and C for $n = 25$ and $t = 0$. Therefore, we will investigate the maximal possible benefit we can get for this configuration in the next section. In the research to adding one or more truck nodes, it was observed that the largest difference was observed between version A and B for $n = 20$ and $t = 2$. Therefore, this will also be investigated in the next section. Lastly, we will research the effects of implementing version C for $n = 30$ and $t = 0$, as this seems to be a threshold value for the working of logical and comb inequalities.

7.2 Extensive results

The extensive analyses performed aim to assess the maximal possible benefit of adding logical and comb inequalities to the branch-and-cut formulation. To this end, we run two versions parallel and compare the time they have spent in the branch-and-cut tree (T_{BNC}). We also terminate the other run when a version has solved the problem instance, and determine the

optimality gap at that point.

7.2.1 Comparing computational times for Version A and C, $t = 0$ and $n = 25$

For this section of the extensive results, we have solved 50 instances with $n = 25$ and $t = 0$ using Version A and C. Figure 22 and 23 depict the results for the analysis.

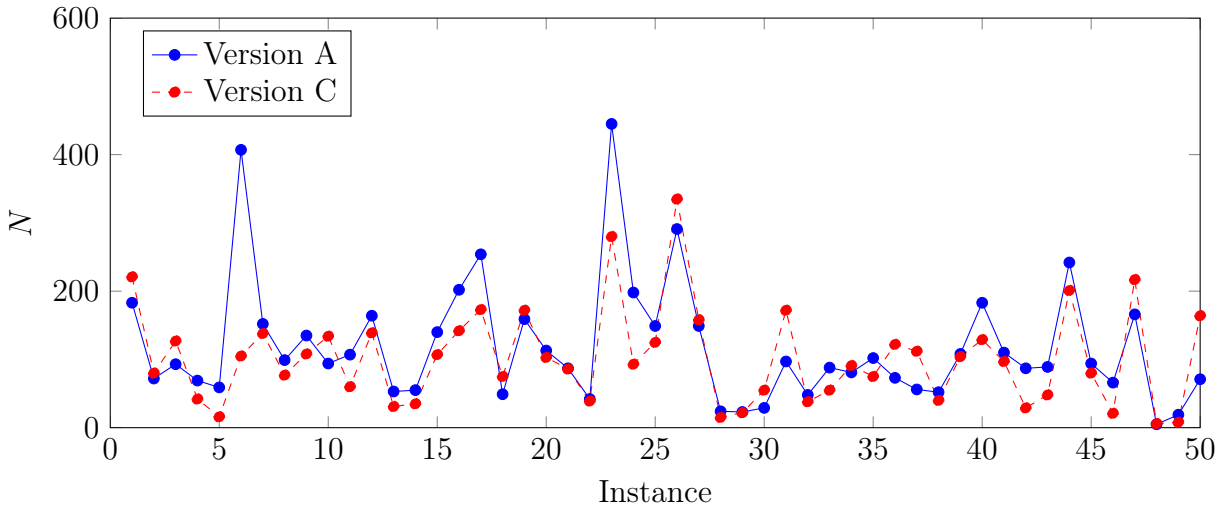


Figure 22: Number of nodes for all instances for $n = 25$

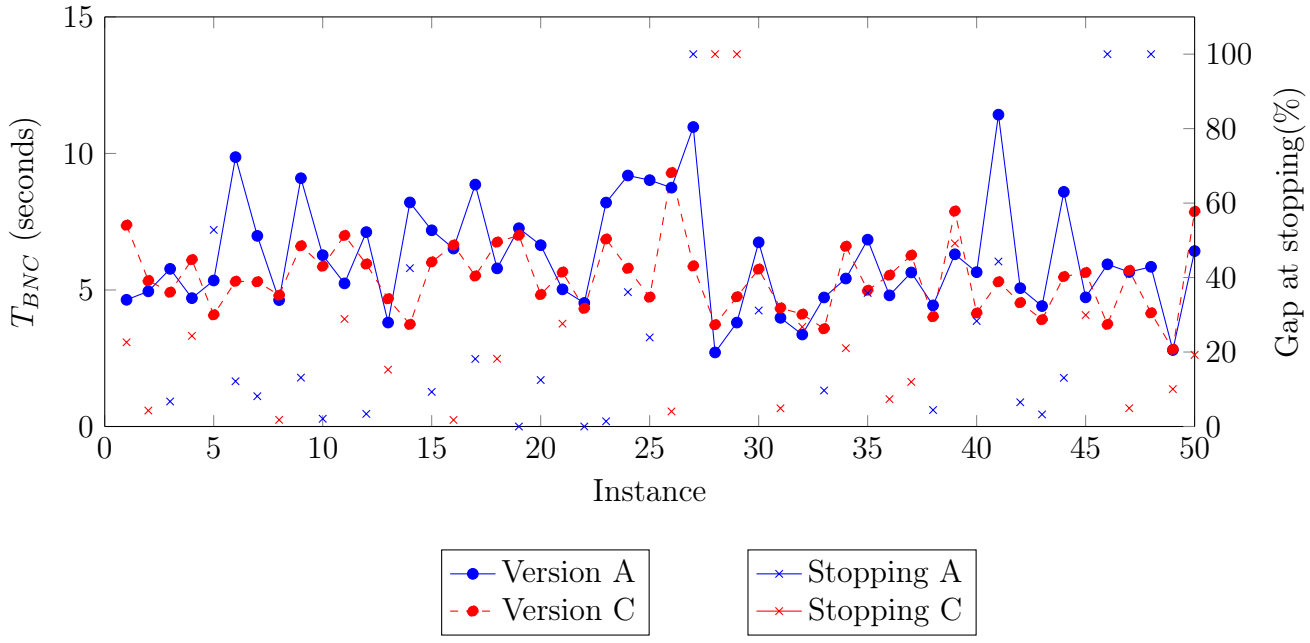


Figure 23: Time and optimality gap for all instances when $t = 0$ and $n = 25$

From Figure 22, we first observe that version A and C generally yield branch-and-cut trees of similar sizes. The average difference observed in the preliminary analysis is mainly caused by the two seemingly difficult instances (6 and 23) where version C outperforms version A. The third difficult instance (26) has a shorter tree with version A, but the difference is much smaller. From Figure 23, it is observed that the time spent in the branch-and-cut tree is very similar. Most differences are just a few seconds. 58.00% of the instances yielded a shorter time spent branching in version C. In 10.34% of these cases, version A did not find an incumbent solution yet. The average gap when stopping version A is 25.67%. For version C, this measure is 24.28%. A more remarkable observation from the two figures is that the computational time spent excluding callbacks does not give an indication of the size of the tree. Instances with obvious peaks in the latter measure, do not have corresponding peaks in T_{BNC}. A possible cause for the lack of relation between these two measures is that the size of the tree does not say anything about the shape of the tree. A large tree can be due to the thorough investigation of a few nodes, going deeply into the branch-and-cut tree, or due to the investigation of many nodes in different directions, mainly expanding the width of the branch-and-cut tree. As the branching strategy is optimized by CPLEX, we do not have insight in how branching evolves in every instance. Moreover, the version and branching strategy affect the time spent in the two subsequent processes of looking for an incumbent solution and converging from an incumbent solution to the optimal solution. These black box methods can cause the occurrence of 100% optimality gaps in Figure 23. For this configuration, we observe that the terminated version is often quite close to the optimal value. This applies to both of the tested versions. On the one hand, this effect means that the logical and comb inequalities work very well for $n = 25$ and $t = 0$. On the other hand, it means that we do not get that much profit from running the two versions in parallel.

7.2.2 Comparing computational times for Version A and C, $t = 0$ and $n = 30$

For this section of the extensive results, we have solved 50 instances with $n = 30$ and $t = 0$ using Version A and C. Figure 24 and 25 depict the results for the analysis.

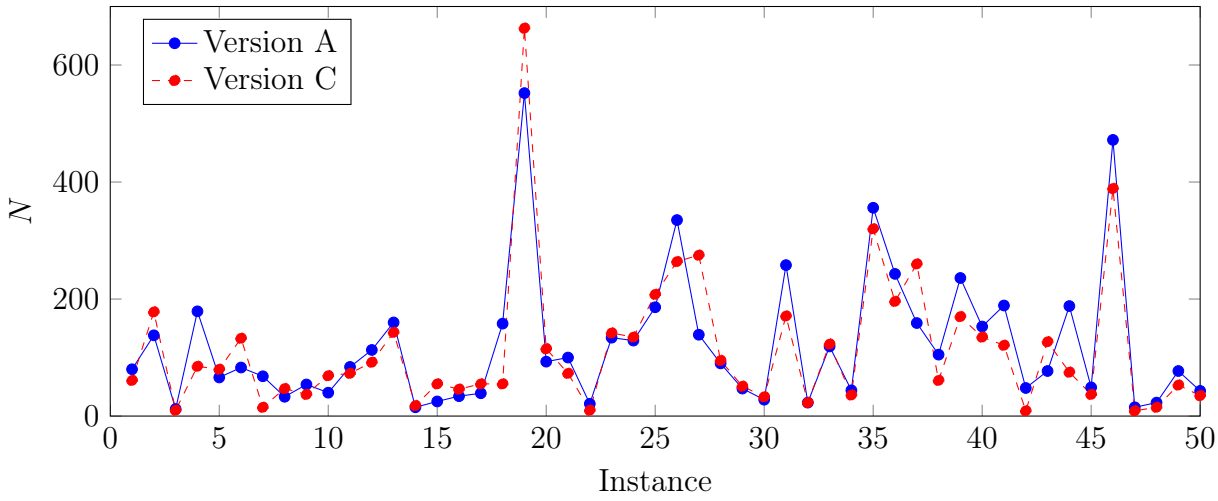


Figure 24: Number of nodes for all instances in version A and C

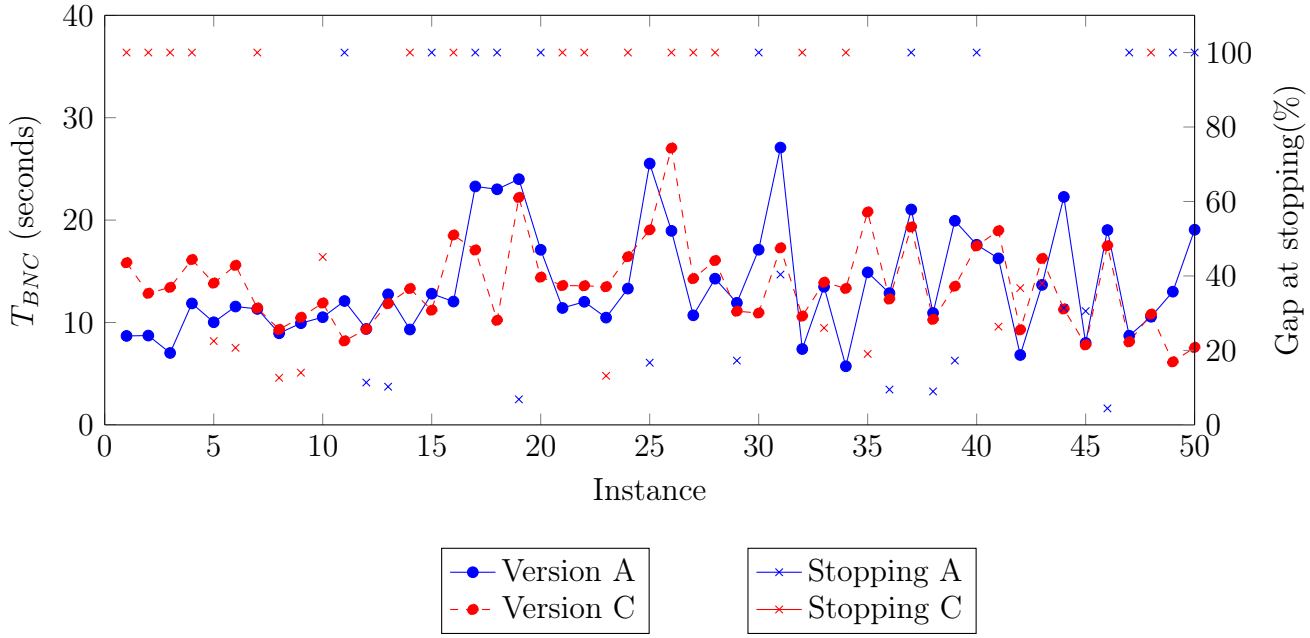


Figure 25: Time in branch-and-cut tree and optimality gap when the other version has solved the problem

As expected, there are no instances for which the total computation time is less using version C compared to version A. More interestingly, Version A spent less computational time (excluding callbacks) in 54% of the instances, compared to Version C. For 32% of the instances, Version C had not found an incumbent solution at the time Version A reached the optimal

solution. The reverse effect, where Version A had not found an incumbent solution at termination, was observed for 22% of the instances. This implies that, in case the two versions are run in parallel all sub-tour and comb inequalities are found immediately, the two versions are comparable. In case one wants to find a tight incumbent bound within a certain time limit and solving two versions in parallel is not possible, version A is preferred. However, we consider the presence of 100% optimality gaps as positive, as this means that parallel solving is useful in this case.

7.2.3 Comparing computational times for versions A and B, $t = 2$, $n = 20$

Figures 26 and 27 depict the results of the final part of the extensive results. It is stressed that we use version B instead of version C in this analysis. Therefore, all observed effects can be contributed to the logical inequalities.

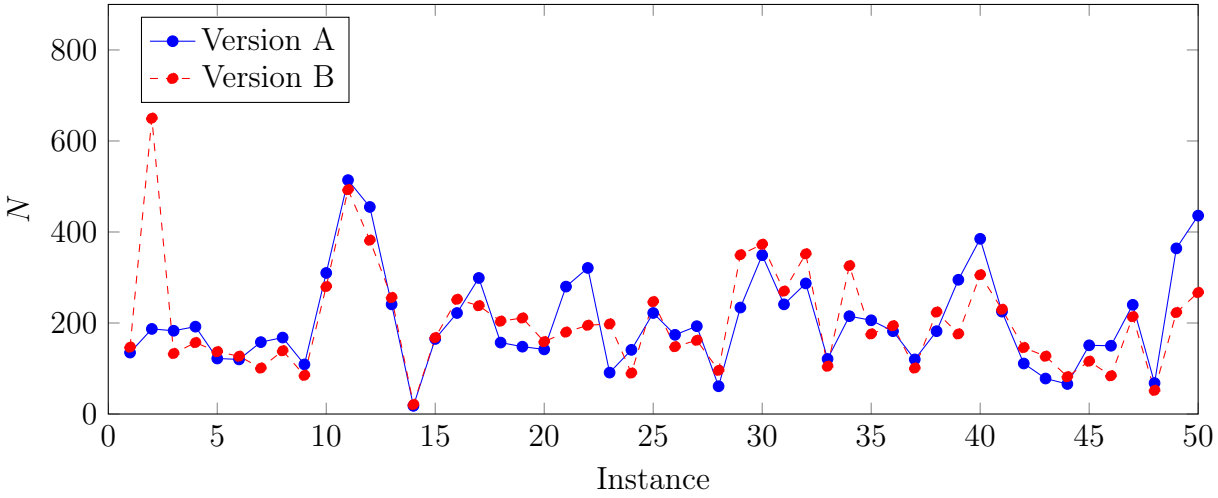


Figure 26: Number of nodes for all instances with $t = 2$, $n = 20$

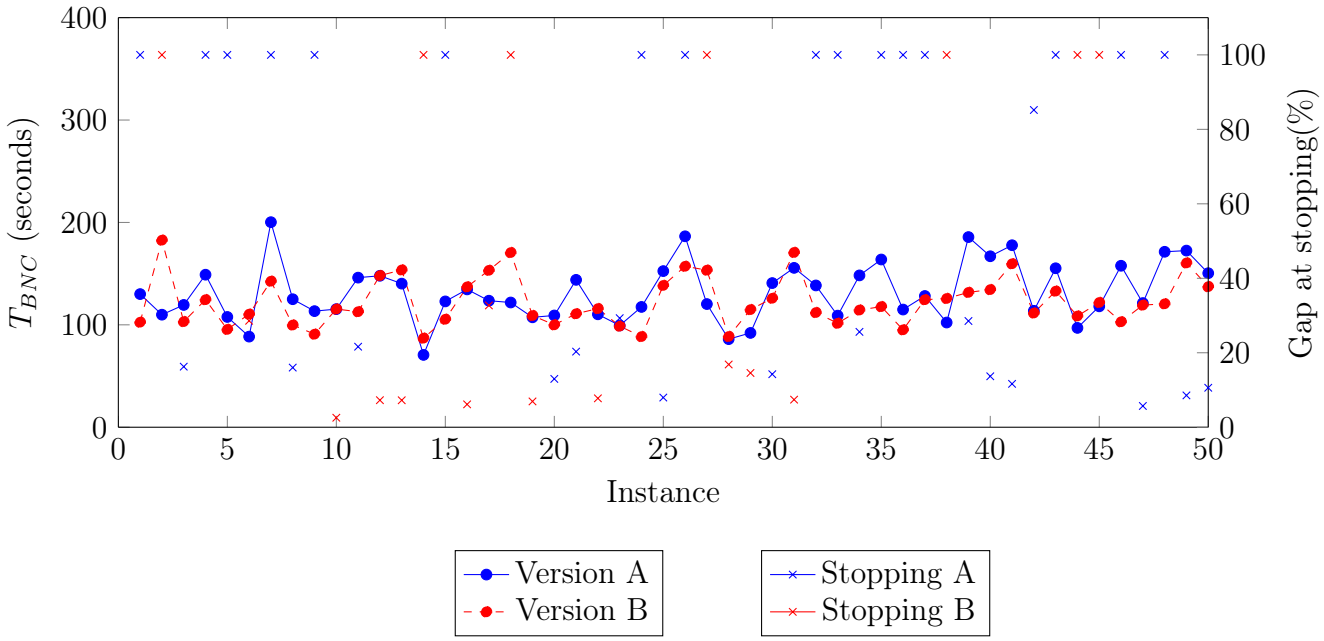


Figure 27: Time and optimality gap for all instances when $t = 2$ and $n = 20$

The most remarkable observation in Figure 26 is the increase in number of nodes when using version B for instance 2. Apparently, the logical inequalities cause the CPLEX algorithm to inspect a suboptimal branch of the tree in detail. The number of nodes in the branch-and-cut tree for other versions does not yield much difference between version A and B. The same applies to the time spent in the branch-and-cut tree. It appears that the increase in number of truck nodes, and therefore the number of operations, affects the time spent in the branch-and-cut tree over all instances. Most instances spend over 100 seconds branching. For this particular configuration, it seems that the optimality gaps are either very high or very low. This is probably due to the relatively small instance size. Whenever an incumbent solution is found, it is immediately quite close to the optimal solution. In this analysis, version B spent less time in the branch-and-cut tree for 66.00% of the instances. In 51.52% of these instances, version A had not found an incumbent solution yet. Vice versa, this measure is 27.59%. As no version is clearly dominant and the time spent in the branch-and-cut tree is quite high, we conclude that running two versions parallel is efficient for this kind of instances. Of course, this still assumes that comb inequalities are found immediately.

8 Conclusion

In this thesis, we have introduced logical and comb inequalities for the Traveling Salesman Problem with assistance of a Drone. The logical inequalities could be derived directly from existing literature on similar problems, such as the Prize Collecting Traveling Salesman Problem and the Generalized Traveling Salesman Problem. To derive valid comb inequalities, we first used the original definition from the classic Traveling Salesman Problem. As the implementation of this type of inequalities was very difficult, we have derived an alternative formulation that corresponds to comb inequalities used in branch-and-cut algorithms for related problems. Next, exact and heuristic separation algorithms were developed to identify the valid inequalities during branch-and-cut algorithm. For the logical inequalities, complete enumeration is a fast and exact separation algorithm. For the comb inequalities we use both the heuristic algorithm by Padberg and Rinaldi, that is based on the block decomposition of the fractional subgraph, as well as an exact algorithm consisting of solving a Mixed Integer Program. To improve the separation of comb inequalities, we also fill a cut pool with variants of earlier found combs and transform invalid combs to valid inequalities using inventive heuristics. All separation algorithms are called in the branch-and-cut algorithm based on a priority list. From a preliminary analysis, we observed that the branch-and-cut algorithm is much faster than the known solution methods, but the proposed inequalities do not always yield an improvement. The addition of the proposed valid inequalities can cause both an increase and a decrease in the number of nodes of the branch-and-cut tree. The logical inequalities prove to be very strong and easily separated. As the heuristic separation finds very few combs and the exact separation procedure takes much time in total, the proposed branch-and-cut algorithm is not competitive for other solving procedures regarding computational time. It does, however, solve problem instances with up to 40 customer nodes to optimality and for instances with less than 30 customer nodes, the tree is often smaller when using the logical and comb cuts. In general, the branch-and-cut algorithm works best when CPLEX is not allowed to delete constraints it deems redundant. In the extensive analysis, the maximal benefit of three configurations were investigated. From these results, it can be concluded that the proposed cuts can be competitive in case the inequalities can be found

faster. We have observed that logical and comb inequalities work well for difficult instances with 25 customer nodes and no truck nodes included in the operations. Running two versions in parallel was especially useful for instances with 30 customers and no truck nodes allowed in the operations, and instances with 25 customers and 2 truck nodes allowed.

9 Further research

Even though the research for this thesis was conducted with great care and effort, there are several limitations to it which can be studied in future research. First, the focus of this research has been on the derivation and the effect of comb inequalities. The logical inequalities from the aforementioned literature formed a well-fitting and strong addition to the comb inequalities. However, it is very well possible that even stronger inequalities can be derived in further research. Examples of this include the families of source-destination and clique tree inequalities derived by Balas (1995) for the Prize Collecting Traveling Salesman Problem and the path inequalities for the classic Traveling Salesman Problem. When allowing the drone to pick up parcels *on the road*, one could use the co-circuit cuts, satellite depot cuts and depot degree constraints used in Belenguer et al. (2016). Moreover, there are many other possible heuristic separation algorithms that might find more valid inequalities than the Padberg and Rinaldi (1990b) algorithm and match up to its speed. From the reviewed literature, other existing heuristic algorithms are the Padberg and Grotschel (1985) heuristic and the blossom finding heuristic proposed by Fischetti et al. (1998). As in the classic Traveling Salesman Problem, one could also consider shrinking techniques and cut metamorphoses to find more valid inequalities. Lastly, this research does not consider improvement heuristics, which are frequently used in branch-and-cut algorithms for related combinatorial problems. This research uses the built-in heuristics of CPLEX to improve the lower bound in the root node and incumbent solutions throughout the algorithm. It could be useful to implement self-built or existing improvement heuristics from literature to further speed up the algorithm in terms of nodes of the branch-and-cut tree and computational time. At this moment, another research is conducted to implementing pricing operations, yielding a branch-and-cut-and-price algorithm. This extension could overcome the limitation of allowed truck nodes in this

research, as we have only covered cases with up to 5 allowed truck nodes. Allowing more truck nodes and adding these operation variables with a pricing mechanism will yield a solution that is guaranteed to be optimal in every aspect.

10 Bibliography

N. Agatz, P. Bouman, and M. Schmidt. Optimization Approaches for the Traveling Salesman Problem with Drone. *Transportation Science*, pages 1–40, 2018.

D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Finding Cuts in the TSP. Technical report, DIMACS, 1993.

D. Applegate, R. Bixby, V. Chvátal, and W. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2011.

E. Balas. The prize collecting traveling salesman problem. *Networks*, 19:621–636, 1989.

E. Balas. The prize Collecting Traveling Salesman Problem: II. Polyhedral Results. *Networks*, 25:199–216, 1995.

E. Balas and M. Fischetti. A lifting procedure for the asymmetric traveling salesman polytope and a large class of facets new. *Mathematical Programming*, 58:325–352, 1993.

J.M. Belenguer, E. Benavent, A. Martínez, C. Prins, C. Prodhon, and J.G. Villegas. A Branch-and-Cut Algorithm for the Single Truck and Trailer Routing Problem with Satellite Depots. *Transportation Science*, 50(2):735–749, 2016.

A. Ben-Dor and B. Chor. On constructing radiation hybrid maps. *Journal of Computational Biology*, 4:517–533, 1997.

J. Bérubé, M. Gendreau, and J. Potvin. A branch-and-cut algorithm for the undirected prize collecting traveling salesman problem. *Networks*, 54(1):56–67, 2009.

D. Bienstock, M.X. Goemans, D. Simchi-Levi, and D. Williamson. A note on the prize collecting traveling salesman problem. *Mathematical Programming*, 59:413–420, 1993.

P. Bouman, N. Agatz, and M. Schmidt. Dynamic Programming Approaches for the Traveling Salesman Problem with Drone. 2017.

J.F. Campbell, D.C. Sweeney II, and J. Zhang. Strategic Design for Delivery with Trucks and Drones. Technical report, 2017.

J.G. Carlsson and S. Song. Coordinated logistics with a truck and a drone. *Management Science*, 2017.

C. Cheng, Y. Adulyasak, and Louis-Martin Rousseau. Formulations and exact algorithms for drone routing problem. 2018.

B.N. Coelho, V.N. Coelho, I.M. Coelho, L.S. Ochi, D. Zuidema, M.S.F. Lima, A.R. da Costa, et al. A multi-objective green uav routing problem. *Computers & Operations Research*, 88:306–315, 2017.

H. Crowder and M.W. Padberg. Solving Large-Scale Symmetric Travelling Salesman Problems to Optimality. *Management Science*, 26(5):495–509, 1980.

R. Daknama and E. Kraus. Vehicle routing with drones. *arXiv preprint arXiv:1705.06431*, 2017.

G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. Technical report, RAND Corporation, Santa Monica, California, USA, 1954.

M. Dell'Amico, F. Maffioli, and P. Varbrand. On Prize-collecting Tours and the Asymmetric Travelling Salesman Problem. *International Transactions in Operations Research*, 2(3): 297–308, 1995.

L. Di Puglia Pugliese and F. Guerriero. Last-mile deliveries by using drones and classical vehicles, 2017.

K. Dorling, J. Heinrichs, G.G. Messier, and S. Magierowski. Vehicle routing problems for drone delivery. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(1): 70–85, 2017.

D. Feillet, P. Dejax, and M. Gendreau. Traveling Salesman Problems with Profits. *Transportation Science*, 39(2):188–205, 2005.

S.M. Ferrandez, T. Harbison, T. Weber, R. Sturges, and R. Rich. Optimization of a truck-drone in tandem delivery network using k-means and genetic algorithm. *Journal of Industrial Engineering and Management*, 9(2):374–388, 2016.

M. Fischetti, J.J.S. González, and P. Toth. A Branch-and-Cut Algorithm for the Symmetric Generalized Traveling Salesman Problem. *Operations Research*, 45(3):378–394, 1997.

M. Fischetti, J.J.S. Gonzalez, and P. Toth. Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, 10(2):133–148, 1998.

M. Flood. The Traveling-Salesman problem. *Management Science*, 1(2):61–75, 1955.

J.C. Freitas and P.H.V. Penna. A Variable Neighborhood Search for Flying Sidekick Traveling Salesman Problem. (December 2013), 2018.

M. Gendreau, G. Laporte, and F. Semet. The Covering Tour Problem. *Operations Research*, 45(4):568–576, 1997.

M. Gendreau, G. Laporte, and F. Semet. A Branch-and-Cut Algorithm for the Undirected Selective Traveling Salesman Problem. *Networks*, 32:263–273, 1998.

B. Golden, Z. Naji-Azimi, S. Raghavan, M. Salari, and P. Toth. The Generalized Covering Salesman Problem. *INFORMS, Journal of Computing*, 24(4):534–553, 2012.

M. Grotschel and O. Holland. Solution of Large Scale Symmetric Travelling Salesman Problem. *Mathematical Programming*, 51:141–202, 1991.

Q.M. Ha, Y. Deville, Q.D. Pham, and M.H. Hà. Heuristic methods for the traveling salesman problem with drone. Technical report, ICTEAM/INGI/EPL, 2015.

M. Held and R.M. Karp. A Dynamic Programming Approach to Sequencing Problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.

S. Hong. *A Linear Programming Approach for the Traveling Salesman Problem*. PhD thesis, Johns Hopkins University, Baltimore, Maryland, USA, 1972.

T. Keeney. Amazon Drones Could Deliver a Package in Under Thirty Minutes for One Dollar. `https://ark-invest.com/research/amazon-drone-delivery#fn-5091-4`, 2015. [Online; accessed 23-April-2018].

J.B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

H.L. Lee, Y. Chen, B. Gillai, and S. Rammohan. *Technological Disruption and Innovation in Last-Mile Delivery. White Paper.* Stanford Graduate School of Business, 2016.

A.C. Leifer and M.B. Rosenwein. Strong linear programming relaxations for the orienteering problem. *European Journal of Operational Research*, 73:517–523, 1994.

S. Lin and B.W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.

M. Marinelli, L. Caggiani, M. Ottomanelli, and M. Dell'Orco. En route truck–drone parcel delivery for optimal vehicle routing strategies. *IET Intelligent Transport Systems*, 12(4): 253–261, 2017.

N. Mathew, S.L. Smith, and S.L. Waslander. Planning paths for package delivery in heterogeneous multirobot teams. *IEEE Transactions on Automation Science and Engineering*, 4 (12):1298–1308, 2015.

C.C. Murray and A.G. Chu. The flying sidekick traveling salesman problem : Optimization of drone-assisted parcel delivery. *Transportation Research Part C: Emerging Technologies*, 54:86–109, 2015.

E. Oswald. Here's everything you need to know about Amazon's drone delivery project, Prime Air. `https://www.digitaltrends.com/cool-tech/amazon-prime-air-delivery-drones-history-progress/`, 2017. [Online; accessed 19-April-2018].

M. Othman, M. Shahrizan, A. Shurbevski, and H. Nagamochi. Routing of carrier-vehicle systems with dedicated last-stretch delivery vehicle. 2017.

A. Otto, N. Agatz, J. Campbell, B. Golden, and E. Pesch. Optimization approaches for civil applications of unmanned aerial vehicles (uavs) or aerial drones: A survey. *Networks*, 2018.

M. Padberg and G. Rinaldi. An efficient algorithm for the minimum capacity cut problem. *Mathematical Programming*, 47(1-3):19–36, 1990a.

M. Padberg and G. Rinaldi. Facet Identification for the Symmetric Traveling Salesman Polytope. *Mathematical Programming*, 47:219–257, 1990b.

M.W. Padberg and M. Grotschel. Polyhedral Computations. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnoy Kan, and D.B. Shmoys, editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, chapter 9, pages 307–360. Wiley, New York, 1985.

M.W. Padberg and M.R. Rao. Odd Minimum Cut-Sets and b-Matchings. *Mathematics of Operations Research*, 7(1):67–80, 1982.

M.W. Padberg and G. Rinaldi. A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems. 33(1):60–100, 1991.

J. Park, S. Kim, and K. Suh. A comparative analysis of the environmental benefits of drone-based delivery services in urban and rural areas. *Sustainability*, 10(3):888, 2018.

S. Poikonen, X. Wang, and B. Golden. The vehicle routing problem with drones: Extended models and connections. *Networks*, 70(1):34–43, 2017.

A. Ponza. Optimization of drone-assisted parcel delivery. 2016.

J. Scott and C. Scott. Drone delivery models for healthcare. 2017.

C. Snow. Drone Delivery: By The Numbers. `http://droneanalyst.com/2014/10/02/drone-delivery-numbers`, 2014. [Online; accessed 23-April-2018].

M.W. Ulmer and B.W. Thomas. Same-day delivery with a heterogeneous fleet of drones and vehicles. Technical report, 2017.

UnmannedCargo. Drones going postal – a summary of postal service delivery drone trials. http://unmannedcargo.org/drones-going-postal-summary-postal-service-delivery-drone-trials/, 2016. [Online; accessed 11-April-2018].

UPS. UPS Tests Residential Delivery Via Drone Launched From atop Package Car. https://pressroom.ups.com/pressroom/ContentDetailsViewer.page?ConceptType=PressReleases&id=1487687844847-162, 2017. [Online; accessed 23-April-2018].

V. Vorotnikov, I. Gumenyuk, and P. Pozdniakov. Planning the flight routes of the unmanned aerial vehicle by solving the travelling salesman problem. *Technology audit and production reserves*, 4(2 (36)):44–49, 2017.

D. Wang. The economics of drone delivery. https://www.flexport.com/blog/drone-delivery-economics/, 2015. [Online; accessed 19-April-2018].

X. Wang, S. Poikonen, and B. Golden. The vehicle routing problem with drones: several worst-case results. *Optimization Letters*, 11(4):679–697, 2017.

A Formulations

A.1 TSP formulation

$$\min \quad \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{55}$$

$$\text{s.t.} \quad \sum_{j\in V\setminus\{i\}} x_{ij} = 1 \qquad \forall i \in V \tag{56}$$

$$\sum_{i\in V\setminus\{j\}} x_{ij} = 1 \qquad \forall j \in V \tag{57}$$

$$\sum(x_{ij} : (i,j) \text{ has one end in } S \text{ and one end not in } S) \geq 2 \quad \forall S \subset V, S \neq \emptyset \quad (58)$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in E \quad (59)$$

A.2 PCTSP formulation

$$\min \quad \sum_{i \in V} \sum_{j \in V \setminus \{i\}} c_{ij} x_{ij} + \sum_{i \in V} c_i y_i \quad (60)$$

$$\text{s.t.} \quad \sum_{j \in N \setminus \{i\}} x_{ij} + y_i = 1 \quad \forall i \in V \quad (61)$$

$$\sum_{i \in V \setminus \{j\}} x_{ij} + y_j = 1 \quad \forall j \in V \quad (62)$$

$$\sum_{i \in V} w_i y_i \leq U \quad (63)$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in E \quad (64)$$

$$y_i \in \{0,1\} \quad \forall i \in V \quad (65)$$

$$\text{Subgraph } G_L(x,y) \text{ has one cycle of length } k \geq 2 \quad (66)$$

$$y_i = 1 \text{ for all nodes } i \text{ not included in the cycle of } G_L \quad (67)$$

B Numeric results

B.1 Preliminary analysis

The tables in this section give averages of several measures for the instances used in the preliminary results. In Tables 10 and 11, the number of customer nodes is set out horizontally and the number of truck only nodes that is allowed is set out vertically for every version tested. N represents the average number of nodes in the branch-and-cut tree, T represents the average computational time in the branch-and-cut part of the algorithm in seconds, and G is the average gap between the lower bound in the root node and the optimal solution in percentage points. It is emphasized that T does not include the time spent in the callback function, as this would not yield a fair comparison between the different versions.

Table 10: Results preliminary analysis, averages of 30 instances (I)

Version A															
n	15			20			25			30			40		
t	N	U	G	N	U	G	N	U	G	N	U	G	N	U	G
0	39.20	123.47	12.95	70.57	244.20	12.50	128.67	401.50	11.94	113.20	468.93	7.84	369.73	1068.60	8.56
1	65.23	156.17	16.77	169.37	294.23	16.38	277.00	484.73	16.78	758.71	896.03	17.85			
2	82.87	168.07	17.60	211.40	318.07	18.60	411.90	541.10	18.25						
3	91.17	160.27	18.95	212.30	307.43	18.52									
4	98.43	163.57	18.96	231.13	314.93	18.46									
5	91.57	166.40	18.96												
Version B															
0	33.77	128.23	14.24	62.13	219.67	11.71	107.33	334.40	11.13	106.17	412.30	8.39	501.17	875.90	10.41
1	73.27	151.90	16.87	152.13	282.83	17.05	305.97	446.67	17.21	719.13	‘678.87	17.74			
2	77.33	150.33	17.81	200.63	277.60	18.57	401.40	508.23	18.23						
3	88.97	149.33	18.94	234.13	291.43	18.50									
4	90.53	151.10	18.95	235.97	296.50	18.44									
5	98.73	148.93	18.95												
Version C															
0	33.77	128.50	12.49	63.30	223.23	11.71	102.47	324.40	11.13	107.03	428.50	8.39	*	*	*
1	71.97	152.17	16.87	152.77	282.80	17.05	297.03	449.20	17.21	*	*	*			
2	77.33	150.33	17.81	200.20	278.00	18.57	*	*	*						

Table 11: Results preliminary analysis, averages of 30 instances (II)

n	15			20			25			30			40		
t	N	U	G	N	U	G	N	U	G	N	U	G	N	U	G
3	89.03	151.10	18.94	234.13	291.73	18.50									
4	90.53	151.10	18.95	235.80	296.40	18.44									
5	98.73	148.93	18.95												
Version D															
0	33.70	128.33	12.49	62.30	220.47	11.71	109.57	326.63	11.13	104.40	443.43	8.39	*	*	*
1	71.90	152.30	16.87	153.00	283.03	17.05	301.37	451.30	17.21	*	*	*			
2	77.33	150.33	17.81	200.63	277.60	18.57	*	*	*						
3	88.97	151.07	18.94	234.13	291.73	18.50									
4	90.53	151.10	18.95	236.03	296.27	18.44									
5	98.73	148.93	18.95												
Version E															
0	33.73	128.43	12.49	62.17	220.93	11.71	104.37	326.03	11.13	104.13	443.87	8.39	*	*	*
1	72.00	152.33	16.87	152.47	283.53	17.05	300.47	450.17	17.21	*	*	*			
2	77.33	150.33	17.81	200.63	277.60	18.57	*	*	*						
3	88.97	151.07	18.94	234.13	291.60	18.50									
4	90.53	151.10	18.95	236.03	296.27	18.44									
5	98.73	148.93	18.95												

Table 12: Results preliminary analysis, averages of 30 instances (III)

Version F															
n	15			20			25			30			40		
t	N	U	G	N	U	G	N	U	G	N	U	G	N	U	G
0	33.77	128.23	14.24	62.13	219.67	11.71	107.17	334.23	11.13	106.17	412.30	8.39	*	*	*
1	73.27	151.90	16.87	152.13	282.83	17.05	305.97	446.67	17.21	*	*	*			
2	77.33	150.33	17.81	200.63	277.60	18.57	*	*	*						
3	88.97	151.07	18.94	234.13	291.43	18.50									
4	90.53	151.10	18.95	235.97	296.50	18.44									
5	98.73	148.93	18.95												

Table 13: Results for instance sizes in $[32, 38]$, $t = 0$, averages of 30 instances

n	32		34		36		38	
Version	N	U	N	U	N	U	N	U
A	128.10	528.10	153.50	636.97	181.67	746.27	263.63	878.60
B	134.13	522.97	157.40	541.37	229.37	663.10	330.80	755.97

B.2 Extensive analysis

This section covers the numeric results of the figures in Section 7.2. The number of nodes in the branch-and-cut tree (N), time spent in the branch-and-cut tree (T_{BNC}), gap in root node (G) and gap when stopping the other version (G_{stop}) are reported.

Table 14: Results comparing versions A and C, $n = 25$ and $t = 0$ (I)

Instance	N_A	N_C	$T_{BNC,A}$(s)	$T_{BNC,C}$(s)	G_A(%)	G_C(%)	G_{stop}(%)
1	183	221	4.64	7.374	11.54	11.55	22.62
2	72	80	4.95	5.344	12.43	11.66	4.28
3	93	127	5.773	4.919	11.12	11.16	6.72
4	69	42	4.701	6.111	11.03	7.76	24.34
5	59	16	5.347	4.09	13.09	10.52	52.80
6	407	105	9.865	5.318	14.83	10.22	12.15
7	152	138	6.982	5.301	9.29	9.53	8.13
8	99	77	4.63	4.812	9.65	9.14	1.79
9	135	108	9.093	6.618	10.65	10.03	13.13
10	94	134	6.274	5.871	12.78	12.32	2.08
11	107	60	5.241	6.997	12.38	10.17	28.87
12	164	139	7.119	5.95	13.28	12.56	3.39
13	53	31	3.807	4.678	11.65	10.37	15.27
14	55	35	8.206	3.739	9.43	9.20	42.52
15	140	107	7.186	6.026	9.90	11.85	9.30
16	202	142	6.518	6.647	14.25	11.80	1.76
17	254	173	8.859	5.513	14.15	13.37	18.15

Table 15: Results comparing versions A and C, $n = 25$ and $t = 0$ (II)

Instance	N_A	N_C	$T_{BNC,A}$(s)	$T_{BNC,C}$(s)	G_A(%)	G_C(%)	G_{stop}(%)
18	49	75	5.791	6.753	14.29	14.27	18.19
19	159	172	7.261	7.003	16.60	16.60	0.00
20	113	103	6.641	4.833	12.98	12.68	12.48
21	87	86	5.023	5.655	14.74	14.52	27.62
22	42	39	4.525	4.327	9.03	8.21	0.00
23	445	280	8.201	6.863	11.70	11.92	1.40
24	198	93	9.192	5.793	13.40	12.31	36.09
25	149	125	9.022	4.737	20.95	20.49	23.92
26	291	335	8.747	9.299	14.97	15.82	4.05
27	149	158	10.968	5.88	14.26	13.73	100.00
28	24	15	2.712	3.727	8.77	7.43	100.00
29	23	22	3.806	4.75	9.11	7.43	100.00
30	29	55	6.745	5.763	12.74	10.44	31.15
31	97	172	3.98	4.34	10.24	10.24	4.89
32	48	38	3.369	4.114	8.91	8.90	26.77
33	88	55	4.725	3.582	10.72	9.34	9.67
34	81	91	5.419	6.6	12.02	12.08	21.03
35	102	75	6.842	4.987	5.13	5.13	35.88
36	73	122	4.803	5.543	7.91	8.43	7.36
37	56	112	5.641	6.275	15.32	16.32	11.98
38	52	40	4.436	4.022	10.30	8.45	4.42
39	108	104	6.307	7.89	14.33	14.57	49.18
40	183	129	5.652	4.156	8.48	8.05	28.34
41	110	97	11.42	5.3	12.68	10.37	44.31
42	87	29	5.066	4.537	7.42	6.53	6.51
43	89	48	4.41	3.914	9.01	8.20	3.24
44	242	201	8.593	5.49	18.07	16.54	13.07
45	94	80	4.73	5.637	7.11	7.23	29.93
46	66	21	5.94	3.742	10.62	11.55	100.00
47	166	217	5.658	5.714	16.93	16.46	4.92
48	5	6	5.848	4.161	10.05	10.13	100.00

Table 16: Results comparing versions A and C, $n = 25$ and $t = 0$ (III)

Instance	N_A	N_C	$T_{BNC,A}$(s)	$T_{BNC,C}$(s)	G_A(%)	G_C(%)	G_{stop}(%)
49	19	8	2.801	2.828	6.32	4.88	10.04
50	71	164	6.424	7.875	15.01	11.90	19.23

Table 17: Results comparing versions A and C, $n = 30$ and $t = 0$ (I)

Instance	N_A	N_C	$T_{BNC,A}$(s)	$T_{BNC,C}$(s)	G_A(%)	G_C(%)	G_{stop}(%)
1	80	61	8.688	15.838	7.28	6.89	100.00
2	138	178	8.723	12.851	11.11	11.88	100.00
3	12	10	7.024	13.431	3.45	3.85	100.00
4	179	85	11.862	16.145	9.40	9.39	100.00
5	66	80	10.023	13.847	7.66	8.31	22.51
6	83	133	11.559	15.581	6.52	8.44	20.70
7	68	15	11.324	11.410	8.83	6.24	100.00
8	33	47	8.948	9.320	6.92	7.44	12.65
9	54	37	9.931	10.503	6.99	7.76	14.02
10	40	69	10.517	11.901	6.82	6.19	45.09
11	84	73	12.098	8.202	8.96	9.34	100.00
12	113	92	9.371	9.349	9.17	10.17	11.38
13	160	143	12.754	11.835	11.35	11.51	10.26
14	15	18	9.316	13.312	3.65	3.82	100.00
15	25	55	12.812	11.204	5.19	5.02	100.00
16	34	46	12.043	18.539	6.12	10.11	100.00
17	39	55	23.286	17.079	6.44	6.49	100.00
18	158	55	23.010	10.216	13.85	10.96	100.00
19	552	663	23.998	22.215	18.22	20.54	6.86
20	93	115	17.100	14.427	8.30	9.13	100.00
21	100	73	11.417	13.606	6.60	7.26	100.00
22	21	10	12.021	13.585	5.51	4.10	100.00

Table 18: Results comparing versions A and C, $n = 30$ and $t = 0$ (II)

Instance	N_A	N_C	$T_{BNC,A}$(s)	$T_{BNC,C}$(s)	G_A(%)	G_C(%)	G_{stop}(%)
23	134	142	10.476	13.495	6.08	5.63	13.18
24	129	135	13.308	16.426	8.37	11.32	100.00
25	186	208	25.522	19.044	16.75	15.50	16.71
26	335	264	18.949	27.033	14.57	15.35	100.00
27	139	275	10.700	14.284	7.32	10.33	100.00
28	90	95	14.278	16.053	7.42	10.17	100.00
29	47	51	11.920	11.103	6.47	6.38	17.27
30	28	33	17.107	10.926	5.44	6.28	100.00
31	258	171	27.085	17.284	8.50	7.58	40.38
32	23	23	7.406	10.619	4.33	7.33	100.00
33	119	123	13.491	13.928	10.95	14.20	26.04
34	44	36	5.725	13.328	3.15	5.69	100.00
35	356	320	14.896	20.788	6.73	6.40	19.06
36	243	196	12.862	12.288	6.43	9.93	9.49
37	159	260	21.036	19.326	9.73	11.94	100.00
38	105	61	10.925	10.305	4.63	5.16	8.98
39	236	170	19.933	13.541	11.40	10.10	17.27
40	153	135	17.591	17.474	5.49	4.68	100.00
41	189	121	16.265	18.964	6.81	7.44	26.38
42	48	9	6.825	9.277	4.46	2.92	36.74
43	77	127	13.673	16.250	8.62	10.75	38.01
44	188	75	22.264	11.321	13.02	10.65	31.38
45	49	37	8.004	7.816	4.30	4.59	30.55
46	472	389	19.028	17.497	13.88	14.74	4.43
47	15	9	8.707	8.111	3.69	2.27	100.00
48	23	15	10.560	10.814	5.86	6.55	100.00
49	77	53	13.009	6.156	6.03	8.02	100.00
50	43	35	19.060	7.562	6.31	12.02	100.00

Table 19: Results comparing versions A and B, $n = 20$ and $t = 2$ (I)

Instance	N_A	N_C	$T_{BNC,A}$(s)	$T_{BNC,B}$(s)	G_A(%)	G_B(%)	G_{stop}(%)
1	135	147	130.027	102.594	15.49	15.18	100.00
2	187	650	109.902	182.695	22.00	22.00	100.00
3	183	133	119.43	103.143	17.55	17.55	16.27
4	192	157	148.959	124.562	15.00	15.00	100.00
5	122	137	107.687	95.552	23.91	23.91	100.00
6	120	127	88.491	110.455	13.88	13.78	28.59
7	158	101	200.257	142.609	16.93	16.93	100.00
8	168	139	125.014	99.835	15.94	15.94	16.02
9	109	85	113.381	90.885	17.79	17.79	100.00
10	310	280	115.345	115.385	19.39	19.39	2.55
11	514	493	146.089	112.838	23.40	23.40	21.63
12	455	382	147.903	148.094	22.25	22.25	7.26
13	241	256	140.235	153.622	22.04	22.04	7.24
14	18	21	70.631	86.982	7.49	7.49	100.00
15	165	168	122.878	105.464	15.40	15.40	100.00
16	222	252	134.599	137.102	21.75	21.75	6.14
17	299	238	123.561	153.233	20.84	20.84	32.71
18	157	204	121.812	170.575	26.49	26.45	100.00
19	148	211	107.339	109.046	18.42	18.42	6.93
20	142	159	109.143	99.978	20.76	20.76	12.98
21	280	180	143.929	110.828	18.35	18.35	20.31
22	321	195	110.28	115.86	18.33	18.33	7.76
23	91	198	99.343	98.74	12.75	12.75	29.29
24	141	90	117.484	88.551	17.42	17.42	100.00
25	222	247	152.484	138.497	20.92	20.92	7.97
26	174	148	186.469	157.177	27.41	27.41	100.00
27	193	162	120.293	153.402	21.76	21.76	100.00

Table 20: Results comparing versions A and B, $n = 20$ and $t = 2$ (II)

Instance	N_A	N_C	$T_{BNC,A}$(s)	$T_{BNC,C}$(s)	G_A(%)	G_C(%)	G_{stop}(%)
28	61	96	86.024	88.625	14.99	14.99	16.85
29	234	350	92.098	114.862	20.89	20.89	14.57
30	349	373	140.793	125.962	19.24	19.24	14.24
31	241	270	155.628	170.727	17.02	16.99	7.41
32	287	352	138.472	112.014	22.05	22.05	100.00
33	121	105	108.99	101.529	13.51	13.47	100.00
34	215	326	148.281	114.459	19.59	19.57	25.61
35	206	176	163.8	117.785	10.22	10.20	100.00
36	182	194	114.927	95.142	14.18	14.18	100.00
37	120	101	128.171	124.54	19.91	19.91	100.00
38	182	224	102.254	125.737	12.90	12.90	100.00
39	295	176	185.708	131.689	24.83	24.83	28.53
40	385	306	166.97	134.418	20.41	20.11	13.68
41	225	230	177.746	159.635	23.98	23.98	11.65
42	111	146	113.701	111.539	14.74	14.74	85.20
43	78	127	155.249	132.923	15.93	15.93	100.00
44	66	82	97.01	108.615	14.45	14.45	100.00
45	151	116	118.052	121.844	14.55	14.55	100.00
46	150	84	157.686	102.995	18.93	18.93	100.00
47	240	214	121.243	119.262	23.81	23.81	5.71
48	68	52	171.343	120.533	19.39	19.39	100.00
49	364	223	172.491	160.451	20.88	20.88	8.55
50	436	267	150.428	137.297	20.91	20.91	10.60