Flow formulations for the Time Window Assignment Vehicle Routing Problem

Master Thesis
Econometrics and Management Science

Kevin Dalmeijer[*]

Erasmus School of Economics
Erasmus University Rotterdam

August 2014

Abstract

In this thesis, we consider flow formulations for the Time Window Assignment Vehicle Routing Problem (TWAVRP), the problem of assigning time windows for delivery before demand volume becomes known. Using commercial solver CPLEX as a basis, we develop a branch-and-cut algorithm that is able to solve all test-instances of Spliet and Gabor (2014) to optimality and is highly competitive with the state of the art solution methods. Furthermore, we introduce a novel set of TWAVRP specific valid inequalities, the precedence inequalities, and show that applying them yields a competitive algorithm as well.

Keywords: *TWAVRP, vehicle routing, flow formulation, precedence inequalities*

[*]Studentnumber 344054kd, supervised by Remy Spliet

Contents

1 Introduction

Distributors have always been interested in decreasing their transport costs by creating efficient routes. The literature clearly reflects this desire with decades of research into problems like the Traveling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP), with or without capacity constraints and/or time constraints.

In this thesis, we consider a variant of the VRP with capacity constraints and time window constraints in which customer demand is not deterministic, but randomly drawn from a finite set of scenarios. Furthermore, we are allowed to set the time windows we need to adhere to ourselves, but only within fixed exogenous time windows and only *before* the demand scenario becomes known. The problem of setting time windows and creating routes for each of the scenarios simultaneously is called the Time Window Assignment Vehicle Routing Problem (TWAVRP), and was first introduced by Spliet and Gabor (2014).

1.1 TWAVRP in practice

The TWAVRP occurs naturally when a distributor serves a fixed set of customers on a regular basis. This is for example the case in the distribution networks of retail chains. The retailers have to be supplied on a regular basis, though their exact demand is not known until a short time before the actual delivery takes place.

In practice, especially in retail, it is common to fix the time windows in which delivery takes place for a longer period, for example a full year. One of the arguments to do so, is that it is very convenient for the retailer to know when delivery takes place, so he can make sure there are enough workers available to process the delivery. Spliet and Gabor (2014) found that current practice in setting time windows is to replace the stochastic demand of each customer by its expected demand and then solve a deterministic vehicle routing problem. Afterwards, it is checked at what time each customer is served and this time is approximately used as the middle of the time window. For example, we agreed with store A to set a time window of 2 hours in width. In the optimal planning based on the expected demand, store A is visited at 12h, which implies a time windows from 11h to 13h. When all time windows are set, optimal routes are created for each scenario, adhering to the time windows.

It is important to notice that this procedure in general does *not* lead to the minimum costs solution, as minimizing costs for the expected demand is inherently different from minimizing expected costs for non-deterministic demand. Spliet and Gabor (2014) encountered up to 5% decrease in costs over the current practice when they solved the TWAVRP for their test instances.

1.2 Related problems

The TWAVRP is related to a large number of problems, including (but not limited to) the Vehicle Routing Problem (VRP), Capacitated Vehicle Routing

Problem (CVRP), Vehicle Routing Problem with Time Windows (VRPTW), Traveling Salesman Problem (TSP), Traveling Salesman Problem with Time Windows (TSPTW) and Bin Packing Problem (BPP).

VRP is the problem of constructing vehicle routes such that all customers are visited by exactly one truck, minimizing costs. It is the predecessor of CVRP, VRPTW and TWAVRP. One limitation of VRP is that is assumes that the trucks have unlimited capacity, which is often not appropriate for real-life applications. To include truck capacity, CVRP was introduced. VRPTW generalizes CVRP by also requiring deliveries to be made in a prespecified time windows. These time windows are assumed to be fixed.

TWAVRP is concerned with actually setting the (endogenous) time windows, though they must be set within larger exogenous time windows. The exogenous time windows could be dictated, for example, by the opening hours of the retailer. When customer demand is known up front, one can simply solve a VRPTW with the exogenous time windows, and pick arbitrary time windows that satisfy the solution of the VRPTW. Spliet and Gabor, however, found that in practice it is often the case that the time windows are set before demand is known, hence the introduction of the TWAVRP.

The relation between VRP and TSP is a clear one: constructing a schedule to visit all customers with only one truck is exactly the TSP. This relation has been exploited for creating solution methods and valid inequalities for variants of the VRP, as will be discussed later.

The BPP is the problem of packing a number of items in the minimum number of bins. Each item has a weight, and the total allowed weight per bin is restricted. This problem can be seen as a special case of the CVRP in which the travel costs on the arcs leaving the depot are equal to one, and all other costs are equal to zero, effectively minimizing the number of vehicles. Furthermore, bin packing occurs as a subproblem in some constraints and valid inequalities that will be discussed later.

1.3 State of the art

VRPs turn out to be difficult to solve to optimality. By now, TSPs of thousands of cities have been solved, while VRPs can only be solved to optimality up to about 100 customers (Laporte, 2009). This number should be used with caution as test instances vary across authors and performance on random instances is often worse. It does give a good impression about the complexity of the problems though.

As the TWAVRP includes determining schedules for all scenarios, it is considerably harder than the VRPTW. It is thus no surprise that only random instances up to 25 customers and 3 scenarios could be solved within one hour of computation time (Spliet and Gabor, 2014).

In the next three subsections, I will discuss the methods with which these state of the art results for CVRP, VRPTW and TWAVRP have been obtained.

1.3.1 CVRP

In a recent survey, Baldacci et al. (2012) mention there are three formulations that have been most successful in solving the CVRP. The first one is the 2-index flow formulation, originally introduced by Laporte et al. (1985). This formulation uses binary variables to indicate flows on the arcs of the network graph. To restrict the vehicle capacity, the so called generalized subtour elimination constraints are introduced. These constraints assure that for each subset of customers enough vehicles enter and leave the subset to satisfy demand. Note that determining a tight lowerbound on the necessary number of vehicles is a bin packing problem, which is NP-hard. Luckily we do not have to solve a NP-hard problem for each subset of customers, as the formulation is also valid when the tight lowerbounds are replaced by not necessarily tight, but easier to calculate lowerbounds.

The second formulation is the 2-commodity flow formulation of Baldacci et al. (2004). Inspired by a 2-commodity flow formulation for the TSP, this formulation is a typical example of TSP results being applied to VRP-like problems. In this formulation, two variables are introduced for each arc of the network graph. The first variable represents the current load of the vehicle, while the second keeps track of the empty space. Truck capacity is then constrained by making sure the empty space on the truck stays non-negative. An advantage of this approach is that the number of restrictions is no longer exponential, like with the 2-index flow formulation.

The third formulation is the set partitioning formulation, first presented by Balinski and Quandt (1964). In the set partitioning formulation, each binary variable is associated with a path in the network graph for which the capacity constraint is met. The formulation itself is a simple one, but requires an exponential number of path parameters to be calculated up front. To overcome this problem, column generation approaches have been introduced. An advantage of using the set partitioning formulation is that, when used in a branch-and-bound framework, such formulations usually yield good lower bounds.

The state of the art results that rely on one of the flow formulations make use of branch-and-cut with special branching rules. An overview of valid inequalities can be found in the survey of Laporte (2009). A recent successful branch-and-cut approach can be found in Lysgaard et al. (2004).

As mentioned, the set partitioning approaches rely on column generation and efficiently solving the associated pricing problem. Again, there is extensive use of valid inequalities to strengthen the bounds. Recent succesfull approaches based on the set partitioning formulation can be found in Fukasawa et al. (2006) and in Baldacci et al. (2008).

1.3.2 VRPTW

For the CVRP we see successful applications using both the flow- and the set partitioning formulations. The literature regarding the VRPTW, however, seems to be much more focused on the set partitioning formulation. Baldacci et al. (2012) mention that this set partitioning formulation has given the best results recently.

One study in which a flow formulation has been applied to solve the VRPTW is the study of Bard et al. (2002). Bard et al. use a different flow formulation from the ones previously discussed. The major difference is that the previous formulations in their basic forms all work on an undirected network graph, while they use a directed graph to include both the capacity and the time constraints.

1.3.3 TWAVRP

The TWAVRP has only been introduced recently, and to the best of my knowledge the only research on this specific problem was conducted by Spliet and Gabor (2014) on the TWAVRP and by Spliet and Desaulniers (2012) on the discrete version of the TWAVRP, in which only a finite number of different time window assignments are allowed. Guided by the VRPTW literature, Spliet and Gabor (2014) also made use of the set partitioning formulation to solve the TWAVRP. This method allows for random instances with up to 25 customers and 3 demand scenarios to be solved within one hour.

1.4 Thesis contribution

Considering the difficulty of the TWAVRP, solving instances up to 25 customers and 3 demand scenarios is impressive. From a practical viewpoint, however, 25 customers is not that much. It is thus important to improve on the solution methods for the TWAVRP so larger instances can also be solved within reasonable time.

In this thesis, we will solve the TWAVRP by using flow formulations in a branch-and-cut framework, opposed to the column generation approach taken by Spliet and Gabor (2014). While flow formulations are expected to give worse lower bounds than when using column generation, the decrease in time necessary to obtain these bounds might yield a net decrease in solving time.

In Section 2 we will present a general flow formulation that allows for different ways of modeling time and capacity. The general formulation will be the basis for our branch-and-cut algorithm. Valid inequalities that may speed up computation are discussed in Section 3 after which we will discuss the separation of these cuts in Section 4. In the sections afterwards, we define our branch-and-cut framework and our experiments. This thesis ends with a discussion of the results and the conclusion.

2 General flow formulation

As we have already seen in the previous section, there are many different ways to formulate the TWAVRP, even when we restrict ourselves to 2-index flow formulations. In this section we will set up a general flow formulation. That is, we introduce the variables and constraints that all flow formulations have in common. Before doing so, we will first formally state the TWAVRP.

Define the following graph, sets and parameters:

$V' = \{1, \ldots, n\}$	set of nodes representing the customers
$V = V' \cup \{0, n+1\}$	V' including the starting (0) and ending $(n+1)$ depot
$G = (V, E)$	complete, directed graph
c_{ij}	cost of traveling arc $(i,j) \in E$, $c_{ij} \geq 0$, satisfies triangle inequality, symmetric costs $(c_{ij} = c_{ji})$
t_{ij}	time to travel arc $(i,j) \in E$, $t_{ij} > 0$, satisfies triangle inequality, symmetric travel times $(t_{ij} = t_{ji})$
Q	capacity of a single vehicle, non-negative
s_i	start time exogenous time window of node $i \in V'$
e_i	end time exogenous time window of node $i \in V'$
w_i	width of *endogenous* time window of node $i \in V'$
Ω	set of scenarios, $\lvert\Omega\rvert < \infty$
p_ω	probability of scenario $\omega \in \Omega$, $\sum_{\omega \in \Omega} p_\omega = 1$
d_i^ω	demand at node $i \in V$ under scenario $\omega \in \Omega$, $0 \leq d_i^\omega \leq Q$, $d_0^\omega = d_{n+1}^\omega = 0$

Furthermore, define the decision variables:

y_i	start of endogenous time window of node $i \in V'$
x_{ij}^ω	$\begin{cases} 1 & \text{if arc } (i,j) \in E \text{ is used in scenario } \omega \in \Omega \\ 0 & \text{otherwise} \end{cases}$
t_i^ω	point in time customer $i \in V'$ is serviced, under scenario $\omega \in \Omega$

We define a *route* for scenario $\omega \in \Omega$ to be a pair (P, t) in which P is an elementary path in G from 0 to $n+1$ and t is a vector containing the values t_i^ω for all nodes visited by P, in order. A *feasible route* for scenario $\omega \in \Omega$ is a route for which all of the following statements hold:

1. the sum of the demands for all nodes visited by P is not larger than Q
2. the exogenous time window constraints are met $(s_i \leq t_i^\omega \leq e_i \; \forall i \in V')$
3. t is feasible $(t_j^\omega \geq t_i^\omega + t_{ij} \; \forall \{i, j \in V \mid (i,j) \in E \text{ is used in path } P\})$

The TWAVRP is the problem of determining time windows (y_v) before demand is known. After the demand scenario has been revealed, a selection of feasible routes is chosen in which every customer is visited exactly once and within the previously set time windows. The objective of the TWAVRP is to minimize the expected traveling costs $\sum_{\omega\in\Omega} p_\omega \sum_{(i,j)\in A} c_{ij}x_{ij}$.

This definition leads to the following general flow formulation:

General flow formulation:

$$\min \sum_{\omega\in\Omega} p_\omega \sum_{(i,j)\in E} c_{ij}x^\omega_{ij} \tag{1}$$

subject to

$$\sum_{j\in V'\cup\{n+1\}} x^\omega_{ij} = 1 \ \forall i \in V', \omega \in \Omega \tag{2}$$

$$\sum_{i\in V'\cup\{0\}} x^\omega_{ij} = 1 \ \forall j \in V', \omega \in \Omega \tag{3}$$

$$\sum_{j\in V'\cup\{n+1\}} x^\omega_{0j} = \sum_{j\in V'\cup\{0\}} x^\omega_{j,n+1} \ \forall \omega \in \Omega \tag{4}$$

$$\text{Time-of-service constraints} \tag{5}$$

$$\text{Capacity constraints} \tag{6}$$

$$s_0 \le t^\omega_j - t_{0j} \ \forall j \in V', \omega \in \Omega \tag{7}$$

$$e_0 \ge t^\omega_i + t_{i,n+1} \ \forall i \in V', \omega \in \Omega \tag{8}$$

$$y_i \le t^\omega_i \le y_i + w_i \ \forall i \in V', \omega \in \Omega \tag{9}$$

$$s_i \le y_i \le e_i - w_i \ \forall i \in V' \tag{10}$$

$$x^\omega_{n+1,i} = 0 \ \forall i \in V, \omega \in \Omega \tag{11}$$

$$x^\omega_{0,n+1} = 0 \ \forall \omega \in \Omega \tag{12}$$

$$x^\omega_{ij} \in \mathbb{B} \ \forall (i,j) \in E, \omega \in \Omega \tag{13}$$

$$t^\omega_i \ge 0 \ \forall i \in V', \omega \in \Omega \tag{14}$$

$$y_i \ge 0, \ \forall i \in V' \tag{15}$$

In the objective (1) we minimize the expected traveling costs over all scenarios. This is done by multiplying the costs per scenario with the probability that the scenario occurs. Constraints (2) ensures every customer node has outflow one. Combined with (3) this enforces that every customer is visited exactly once. Constraints (4) make sure that all flow sent into the network reaches the destination depot, that is, the number of trucks to leave the depot is the same as the number of trucks that return to the depot. Note that the previous constraints do not eliminate cycles in G. Explicit cycle elimination constraints are not necessary in the general flow formulation because all time-of-service constraints discussed in this thesis will eliminate cycles if all travel times are strictly positive. I am not aware of any set of time constraints that does *not* implicitly eliminate cycles in this case.

Both sets of constraints (5) and (6) have not yet been specified, as there are multiple possibilities. In the following sections, different options for both the time-of-service constraints and the capacity constraints will be presented. The time-of-service constraints, or time constraints for short, ensure that the time of service at each client is modeled correctly. The capacity constraints ensure the vehicle capacity is respected. For reader convenience, the actual constraints they have to be replaced with, are surrounded by boxes. That is, the pragmatic reader of this thesis can pick a box for both (5) and (6) and end up with a valid formulation.

Constraints (7)-(10) deal with the time windows. (7) and (8) ensure trucks only leave and arrive at the depot within its opening hours. Constraint (9) enforces each client is served within his endogenous time window, while constraint (10) makes sure these endogenous time windows are within the exogenous time windows.

Finally, the constraints (11) and (12) place some natural restrictions on the flow variables, which allow more convenient notation in the upcoming sections. They restrict the use of flows out of the node representing the destination depot (11) and flows between the two depot nodes (12).

2.1 Time-of-service constraints

In this section we discuss various ways to model the time constraints that replace Equation (5).

2.1.1 MTZ-inequalities

A straightforward way to model time is by linearizing the following statement:

$$x_{ij}^{\omega} = 1 \Rightarrow t_j^{\omega} \geq t_i^{\omega} + t_{ij} \;\forall i \in V' \cup \{0\}, j \in V', \omega \in \Omega \tag{16}$$

This statement could be paraphrased as: "if a truck drives from i to j, it arrives at least its travel time later in j". Linearization of if-statements is generally done with the big-M method, yielding:

$$t_j^{\omega} \geq t_i^{\omega} + t_{ij}x_{ij}^{\omega} - M_{ij}(1 - x_{ij}^{\omega}) \;\forall i \in V' \cup \{0\}, j \in V', \omega \in \Omega \tag{17}$$

in which M_{ij} are large positive values such that when $x_{ij}^{\omega} = 0$ the constraint becomes inactive. The inequalities in (17) are often referred to as the MTZ-inequalities, and have been named after Miller, Tucker, and Zemlin (1960).

If we use the exogenous time windows of t_j^{ω} and t_i^{ω} we have $t_j^{\omega} - t_i^{\omega} \geq s_j - e_i$. In our specific case, we can thus use $M_{ij} = e_i - s_j$ without cutting off any feasible solutions. Substituting these values, we have:

Time constraints: MTZ-inequalities

$$t_j^\omega \geq t_i^\omega + t_{ij}x_{ij}^\omega - (e_i - s_j)(1 - x_{ij}^\omega) \; \forall i \in V' \cup \{0\}, j \in V', \omega \in \Omega \qquad (18)$$

2.1.2 Affine combination inequalities

If we combine the logic of the MTZ-inequalities with the fact that every node, except for the depot nodes, has exactly one predecessor node, we can write the current time as an affine combination of all partial predecessors in the following way:

$$t_j^\omega \geq \sum_{i \in V' \cup \{0\}} (t_i^\omega + t_{ij})x_{ij}^\omega \; \forall j \in V', \omega \in \Omega \qquad (19)$$

The correctness of (19) follows from the fact that $\sum_{i \in V' \cup \{0\}} x_{ij}^\omega = 1$ and the integrality of the x-variables.

The problem in using (19) is that the product of t_i^ω and x_{ij}^ω causes the constraints to be non-linear. While this is not generally true for any product of variables, a product involving a binary variable and a bounded continuous variable can always be linearized (Rubin, 2010). Note that x_{ij}^ω is binary and $s_i \leq t_i^\omega \leq e_i$, and hence linearization is possible.

Let us introduce the continuous variable g_{ij}^ω which will be equal to the product of t_i^ω and x_{ij}^ω for every integer solution.

$$g_{ij}^\omega \geq s_i x_{ij}^\omega \qquad (20)$$

$$g_{ij}^\omega \geq t_i^\omega - e_i(1 - x_{ij}^\omega) \qquad (21)$$

$$g_{ij}^\omega \leq e_i x_{ij}^\omega \qquad (22)$$

$$g_{ij}^\omega \leq t_i^\omega - s_i(1 - x_{ij}^\omega) \qquad (23)$$

Equations (20)-(23) are best explained with the help of Figure 1. Figure 1 visualizes the situation in which the value of t_i^ω is fixed at the value t. The dotted line corresponds to the product $t_i^\omega x_{ij}^\omega$, which is the non-linear expression we want to linearize. Equations (20)-(23) are represented by ①-④ respectively. Together they define the feasible region of g_{ij}^ω, the gray area. From the figure it is clear why $x_{ij}^\omega = 0 \Longrightarrow g_{ij}^\omega = 0$ and $x_{ij}^\omega = 1 \Longrightarrow g_{ij}^\omega = t$.

We can now replace (19) by its linear counterpart:

$$t_j^\omega \geq \sum_{i \in V' \cup \{0\}} \left(g_{ij}^\omega + t_{ij}x_{ij}^\omega\right) \; \forall j \in V', \omega \in \Omega \qquad (24)$$

Note that the g-variables only occur in (20)-(23) and (24). It follows that in the LP relaxation, it is optimal to pick g_{ij}^ω as small as possible, to maximizes the slack of (24); picking a larger value for the g-variable only restricts the range of t_j^ω. Hence, it is not necessary to bound g_{ij}^ω from above ((22) and (23)). The full linearization of (19) is then given by:

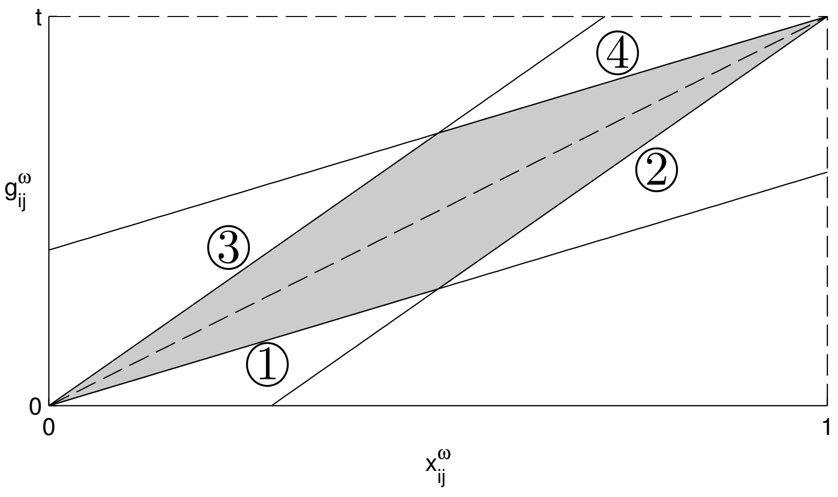

Figure 1: Visualization of (20)-(23)

Time constraints: affine combination predecessors

$$t_j^\omega \geq \sum_{i \in V' \cup \{0\}} \left(g_{ij}^\omega + t_{ij} x_{ij}^\omega \right) \ \forall j \in V', \omega \in \Omega \tag{24}$$

$$g_{ij}^\omega \geq s_i x_{ij}^\omega \ \forall i \in V' \cup \{0\}, \forall j \in V', \omega \in \Omega \tag{20}$$

$$g_{ij}^\omega \geq t_i^\omega - e_i(1 - x_{ij}^\omega) \ \forall i \in V' \cup \{0\}, \forall j \in V', \omega \in \Omega \tag{21}$$

Instead of using an affine combination of the predecessors, we could also use an affine combination of the successors in the following way:

$$t_i^\omega \leq \sum_{j \in V' \cup \{n+1\}} (t_j^\omega - t_{ij}) x_{ij}^\omega \ \forall i \in V', \omega \in \Omega \tag{25}$$

Analogously, we introduce the variable h_{ij}^ω and we get:

Time constraints: affine combination successors

$$t_i^\omega \leq \sum_{j \in V' \cup \{n+1\}} \left(h_{ij}^\omega - t_{ij} x_{ij}^\omega \right) \ \forall i \in V', \omega \in \Omega \tag{26}$$

$$h_{ij}^\omega \leq e_j x_{ij}^\omega \ \forall i \in V', \forall j \in V' \cup \{n+1\}, \omega \in \Omega \tag{27}$$

$$h_{ij}^\omega \leq t_j^\omega - s_j(1 - x_{ij}^\omega) \ \forall i \in V', \forall j \in V' \cup \{n+1\}, \omega \in \Omega \tag{28}$$

Note that g_{ij}^ω differs from h_{ij}^ω as g_{ij}^ω represents $t_i^\omega x_{ij}^\omega$ and h_{ij}^ω represents $t_j^\omega x_{ij}^\omega$.

Both the affine combinations of predecessors and the affine combinations of successors can thus be used at the same time to strengthen the LP relaxation.

2.1.3 Convex hull formulation

Time constraints are naturally linked to logical constraints. We have already seen this in the previous time constraints: the MTZ-inequalities are linearizations of if-conditions and the affine combination inequalities use the logic that every node has exactly one predecessor or successor. Instead of formulating linear constraints directly, it may prove valueble to state the time constraints as logical constraints and apply the theory of disjunctive programming.

That is, we will interpret the flow variables as Boolean variables and model the time constraints as a logical conjunction (logical AND, $\wedge$) of disjunctions (logical OR, $\vee$). There are multiple ways to do so. For example, we can use the fact that every node has one predecessor to get:

$$\bigvee_{i \in V' \cup \{0\}} \left[\begin{array}{l} X_{ij}^{\omega} \\ t_j^{\omega} - t_i^{\omega} \geq t_{ij} \end{array} \right] = true \ \forall j \in V', \omega \in \Omega \tag{29}$$

$$X_{ij}^{\omega} \in \{true, false\} \ \forall i \in V' \cup \{0\}, j \in V', \omega \in \Omega \tag{30}$$

In Equation (29), the block is a conjunction with X_{ij}^{ω}, which represents the arc from i to j in scenario ω, and the associated time constraint. A block thus only has value *true* when the arc is chosen AND the time constraint is met. The OR condition ensures that at least one block is true, and thus at least one predecessor is chosen.

When a linear program contains constraints in this form, the program is known as a linear generalized disjunctive program (LGDP) (Sawaya and Grossmann, 2008). Solution methods for LGDPs vary, but often include reformulation to a mixed integer linear problem. Raman and Grossmann proposed a big-M reformulation for this purpose (Raman and Grossmann, 1994). If this reformulation is applied to (29) and (30), the MTZ-inequalities (18) appear.

Later, Lee and Grossmann (2000) presented a formulation which gives tighter LP bounds, known as the convex hull relaxation. The difference in LP bounds between the big-M formulation and the convex hull formulation is easily understood by examining Figure 2. If we convert a LGDP to a MIP, then each disjunct (a block in (29)) is represented by a feasible region in the MIP. In Figure 2, two such regions are represented by the areas ‘Disjunct 1’ and ‘Disjunct 2’. As a MIP requires a connected feasible region, the disjuncts are ‘connected’ by additional variables and constraints. The convex hull reformulation does this by constructing a convex hull of the disjuncts as the feasible region of the MIP, as shown by the dark grey area. The big-M approach is simpler, but results in a larger feasible region: the light gray area and the dark gray area combined.

We apply the Lee and Grossmann reformulation of (29) by rewriting our problem in the same way as the general LGDP is rewritten in by Sawaya and

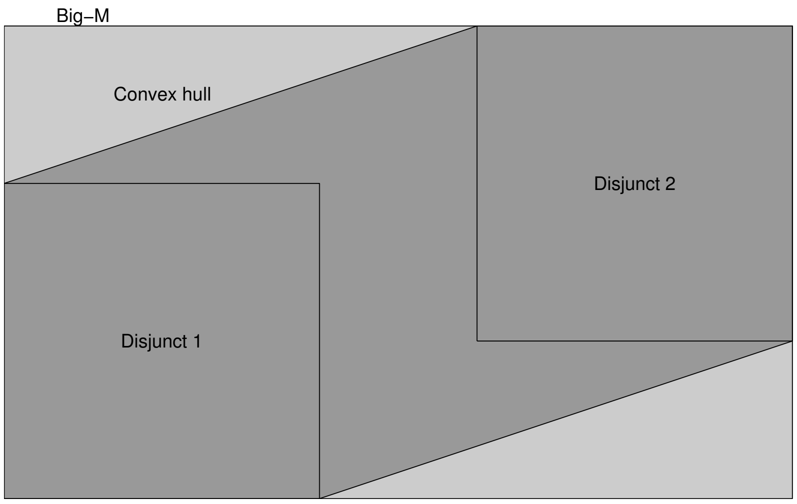

Figure 2: Visualization of the difference between the big-M and the convex hull formulation

Grossmann (2008):

$$t_l^\omega = \sum_{i \in V' \cup \{0\}} v_l^{\omega ij} \ \forall j \in V', l \in V' \cup \{0\}, \omega \in \Omega \tag{31}$$

$$v_j^{\omega ij} - v_i^{\omega ij} \geq t_{ij} x_{ij}^\omega \ \forall i \in V' \cup \{0\}, j \in V', \omega \in \Omega \tag{32}$$

$$s_l x_{ij}^\omega \leq v_l^{\omega ij} \leq e_l x_{ij}^\omega \ \forall i \in V' \cup \{0\}, j \in V', l \in V' \cup \{0\}, \omega \in \Omega \tag{33}$$

$$v_l^{\omega ij} \geq 0 \ \forall i \in V' \cup \{0\}, j \in V', l \in V' \cup \{0\}, \omega \in \Omega \tag{34}$$

While the above is a correct way to model time for the TWAVRP, it still lacks a proper interpretation of *why* it is correct. By close inspection it can be seen that $v_l^{\omega ij}$ allows for the following interpretation: if $x_{ij}^\omega = 1$, $v_l^{\omega ij}$ is equal to the service time of client l in scenario ω. If $x_{ij}^\omega = 0$, $v_l^{\omega ij}$ is also 0.

In other words, for every integer solution, in scenario ω, each subscript l defines a graph in which all arcs used by the vehicles correspond to a v-variable with value t_l^ω. An example is given by Figure 3.

The convex hull formulation allows for a reduction in the number of variables by aggregation, without changing the feasible region in terms of the x- and t-variables. The resulting formulation is presented in Appendix A. The formulation in the appendix is the one that will be implemented.

For now, we will not look further into reformulations based on the LGDP representation, as they are more difficult to apply and do not have a clear interpretation. It has to be mentioned though, there are multiple solution methods available which may also be of use for the TWAVRP. An interesting paper in this respect could be Grossmann and Ruiz (2012).

2.1.4 Improved convex hull formulation

In the previous section, we have introduced the convex hull formulation. When inspecting Figure 3, however, it seems the convex hull formulation is somehow inefficient, as there are multiple v-variables which attain the same non-zero value. For example: there are n v-variables that attain the value t_i^ω. This can be overcome by removing one dimension from the v-variables (the one given by subscript l) and defining the v-variables such that we get the network of Figure 4. That is, we define v_{ij}^ω to equal t_j^ω when $x_{ij}^\omega = 1$ and 0 otherwise.

Formally, we use the variables

$$v_{ij}^\omega \quad \begin{cases} t_j^\omega & \text{if } x_{ij}^\omega = 1, \text{ for } (i,j) \in E \text{ and } \omega \in \Omega \\ 0 & \text{otherwise} \end{cases}$$

This leads to the following formulation:

Time constraints: improved convex hull formulation

$$t_j^\omega = \sum_{i \in V' \cup \{0\}} v_{ij}^\omega \ \forall j \in V', \omega \in \Omega \tag{35}$$

$$\sum_{j \in V' \cup \{n+1\}} v_{ij}^\omega - \sum_{k \in V' \cup \{0\}} v_{ki}^\omega \geq \sum_{j \in V' \cup \{n+1\}} t_{ij} x_{ij}^\omega \ \forall i \in V', \omega \in \Omega \tag{36}$$

$$s_j x_{ij}^\omega \leq v_{ij}^\omega \leq e_j x_{ij}^\omega \ \forall i \in V' \cup \{0\}, j \in V' \cup \{n+1\}, \omega \in \Omega \tag{37}$$

$$v_{ij}^\omega \geq 0 \ \forall i \in V' \cup \{0\}, j \in V' \cup \{n+1\}, \omega \in \Omega \tag{38}$$

Constraint (36) differs from (32) by the summations, which are necessary to determine, respectively, the time of service of the successor of client i, the value of t_i^ω and the travel time from i to its successor. The other constraints are similar to the constraints in the convex hull formulation.

The formulation presented above is not new: it also appears (with slightly different variables) in the literature on the asymmetric traveling salesman problem with time windows (Ascheuer et al., 2001).

2.1.5 Comparison

In the previous sections, we have discussed four distinct ways to model time in the general formulation. The MTZ-inequalities are the simplest in multiple ways. First, the underlying logic is simple. Second, the number of constraints is relatively small compared to the other time constraints (see Table 4). Usually, however, this also results in relatively bad LP bounds.

The affine combination inequalities are based on the stronger logical result that the time of service of a client is later than the affine combination of the times of service of all predecessors or successors. Based on this, one would expect the

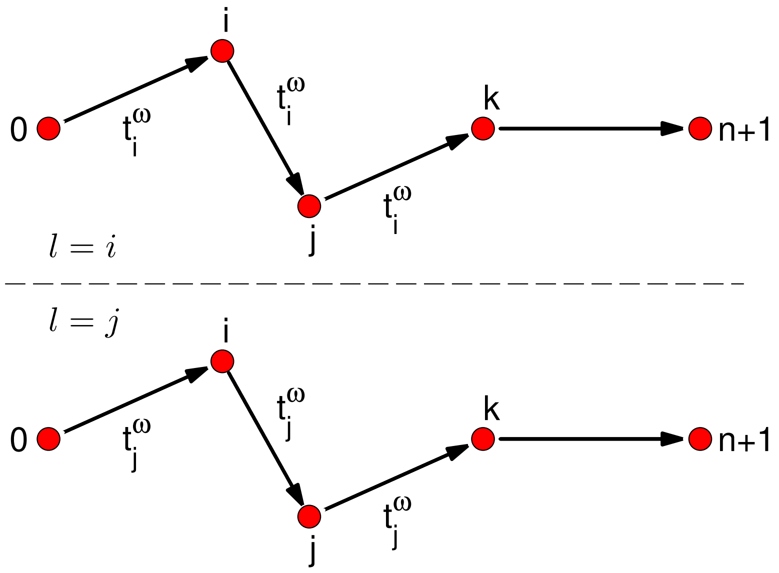

Figure 3: Example of (31)-(34) in scenario ω for a feasible solution of the TWAVRP. The values on the arcs note the values of the associated v-variables. All arcs that are not drawn have a value of 0 for the corresponding v-variables.

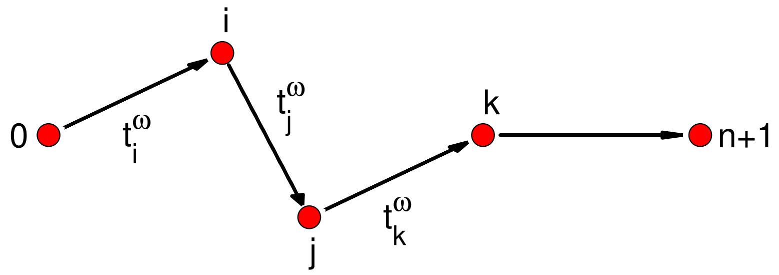

Figure 4: Example of (35)-(38) in scenario ω for a feasible solution of the TWAVRP. The values on the arcs note the values of the associated v-variables. All arcs that are not drawn have a value of 0 for the corresponding v-variables.

affine combination inequalities to be stronger than the MTZ-inequalities. However, the applied linearization may diminish the effect of the stronger logical result. Combined with the slowdown because of the larger number of variables and constraints, it is not certain which formulation will allow for the best solution time when used in a branch-and-cut algorithm.

Taking a detour through disjunctive programming, we also constructed the convex hull formulation to model time. This formulation has the largest number of variables and constraints, but is also assumed to be relatively strong because we use the convex hull of the disjunctions that naturally appear in the TWAVRP. Its effect in a branch-and-cut framework cannot be stated up front, and has to be tested empirically. The improved convex hull formulation reduces the number of variables and constraints of the convex hull formulation considerably, but it is yet uncertain how this affects the strength of the formulation.

Method	Additional variables	Additional constraints
MTZ-inequalities	0	$(n^2+n)\lvert\Omega\rvert$
Affine combination inequalities	$(n^2+n)\lvert\Omega\rvert$	$(2n^2+3n)\lvert\Omega\rvert$
Convex hull formulation	$(3n^2+2n)\lvert\Omega\rvert$	$(8n^2+6n)\lvert\Omega\rvert$
Improved convex hull	$(n^2+2n+1)\lvert\Omega\rvert$	$(n^2+4n+1)\lvert\Omega\rvert$

Table 4: Number of variables and constraints that have to be added to the general flow formulation to model time using one of the discussed methods (excluding ≥ 0 variable bounds)

2.2 Capacity constraints

In this section, we will discuss the various ways to model capacity, that is, to fill in (6) in the general flow formulation. We will start by briefly revisiting the methods of the previous section, and adapt them to model capacity.

2.2.1 MTZ-inequalities

Bard et al. (2002) use the MTZ-inequalities to not only model time, but also truck load. This can be accomplished by defining:

q_i^ω	sufficient truck load to serve up to client $i \in V'$ (inclusive) under scenario ω

For example, if a truck first visits client A and then client B, both with demand 3, then $q_A \geq 3$ and $q_B \geq 6$.

Analogous to Section 2.1.1 we arrive at the following MTZ-inequalities:

Capacity constraints: MTZ-inequalities

$$q_j^\omega \geq q_i^\omega + d_j^\omega x_{ij}^\omega + (d_j^\omega - Q)(1 - x_{ij}^\omega) \ \forall i \in V' \cup \{0\}, j \in V' \cup \{n+1\}, \omega \in \Omega \quad (39)$$

$$d_i^\omega \leq q_i^\omega \leq Q \ \forall i \in V', \omega \in \Omega \quad (40)$$

In which (39) can be seen as the capacity equivalent of the time constraint (17), in which $(d_j^\omega - Q)$ serves as the 'big-M'. Equation (40) ensures truck load never exceeds the capacity.

2.2.2 Affine combination inequalities

Using the same truck load variables, we can construct affine combination inequalities analogous to Section 2.1.2. Because of the similarities, we will not discuss the application of the affine combination inequalities to capacity.

The resulting equations for affine combinations of the predecessors can be found in Appendix B.

2.2.3 (Improved) convex hull formulation

Also both the convex hull formulation and the improved convex hull formulation of Sections 2.1.3 and 2.1.4 can be applied to capacity in an analogous way. We do not present the full formulations.

2.2.4 Generalized subtour elimination constraints

A totally different way of ensuring that the capacity of the trucks is not violated, is by using the generalized subtour elimination constraints. With these constraints, we require for each subset $S \subset V$ that the total number of trucks leaving S is enough to satisfy all demand in S. That is, we add inequalities:

Capacity constraints: generalized subtour elimination constraints

$$\sum_{(i,j)\in(S,V\setminus S)} x_{ij}^\omega \geq \lambda_S^\omega \ \forall S \subset V, \omega \in \Omega \quad (41)$$

with λ_S^ω the minimum number of trucks required to satisfy all demand in S.

There are two disadvantages to these constraints. First, determining λ_S^ω requires solving a bin packing problem, and is thus NP-hard. This problem can be overcome by replacing λ_S^ω by a suitable lower bound, like $\lceil \left(\sum_{i\in S} d_i^\omega\right) / Q \rceil$ (Baldacci et al., 2012). The second disadvantage is that the number of constraints is exponential in the number of customers. Simply adding all constraints is not efficient, but when used in a branch-and-cut procedure, the generalized subtour elimination constraints can be very effective (Baldacci et al. (2012), Spliet and Gabor (2014)).

2.2.5 2-commodity flow

The 2-commodity flow formulation of Baldacci et al. (2004) was already mentioned in the introduction as one of the most successful formulations for solving the CVRP. This makes it interesting to apply the capacity constraints to our model as well. We can directly copy the part of their formulation that handles the capacity constraint, after making two changes:

First, Baldacci et al. (2004) use a fixed number of vehicles. We replace this number by a variable number of vehicles $\sum_{j \in V'} x_{0j}^{\omega}$ for each scenario $\omega \in \Omega$.

Second, Baldacci et al. make use of a binary variable, indicating whether an arc in the *undirected* graph is used or not. We replace this variable by $x_{ij}^{\omega} + x_{ji}^{\omega}$. That is, an arc is used in the undirected graph when either arc between the same nodes in the directed graph is used. Note that when x_{ij}^{ω} and x_{ji}^{ω} are integer, they can not both be 1, as this would create a subtour, which would have been eliminated by the time constraints.

To explain their formulation, Baldacci et al. make use of Figure 5 (adjusted for notation differences). This figure shows a path in scenario ω from the starting depot to the ending depot, visiting clients 8, 2 and 9. This path is shown in the figure by the solid arrows. The reverse path is given by the dashed arrows.

We introduce variables z_{ij}^{ω} for all $(i,j) \in E$, $\omega \in \Omega$. z_{ij}^{ω} thus corresponds to an arc $(i,j) \in E$ in scenario ω. Its meaning, however, depends on whether the arc follows a path from starting depot to ending depot (solid arrow), follows a reverse path (dashed arrow), or is not used by any path (no arrow). In the first case, z_{ij}^{ω} models the total truck load when traveling from i to j. In the second case, z_{ij}^{ω} models the empty space on the truck, when traveling from i to j. If the arc (i,j) is not being used, z_{ij}^{ω} is simply zero. The trick of this formulation is that we can now restrict capacity by simply requiring the empty space on the trucks to be non-negative at all times.

In Figure 5, the vehicle with capacity 15 leaves the starting depot with a total load of 14, which is the total demand of the clients that he will be visiting. This leaves 1 unit of empty space. Client 8 demands 3 units product ($d_8^{\omega} = 3$). During its travel to client 2, the vehicle has a total load of 11 units product and 4 units of empty space. At the end of its trip, the vehicle is empty, so the empty space equals its capacity.

The formulation is given by:

Capacity constraints: 2-commodity flow

$$\sum_{j \in V} (z_{ji}^{\omega} - z_{ij}^{\omega}) = 2d_i^{\omega} \ \forall i \in V', \omega \in \Omega \tag{42}$$

$$\sum_{j \in V'} z_{0j}^{\omega} = \sum_{i \in V'} d_i^{\omega} \ \forall \omega \in \Omega \tag{43}$$

$$\sum_{j\in V'} z_{j0}^{\omega} = \left(\sum_{j\in V'} x_{0j}^{\omega}\right) Q - \sum_{i\in V'} d_i^{\omega} \ \forall \omega \in \Omega \tag{44}$$

$$\sum_{j\in V'} z_{n+1,j}^{\omega} = \left(\sum_{j\in V'} x_{0j}^{\omega}\right) Q \ \forall \omega \in \Omega \tag{45}$$

$$z_{ij}^{\omega} + z_{ji}^{\omega} = \left(x_{ij}^{\omega} + x_{ji}^{\omega}\right) Q \ \forall i \in V, j \in V, \omega \in \Omega \tag{46}$$

$$z_{ij}^{\omega} \geq 0 \ \forall i \in V, j \in V, i \leq j, \omega \in \Omega \tag{47}$$

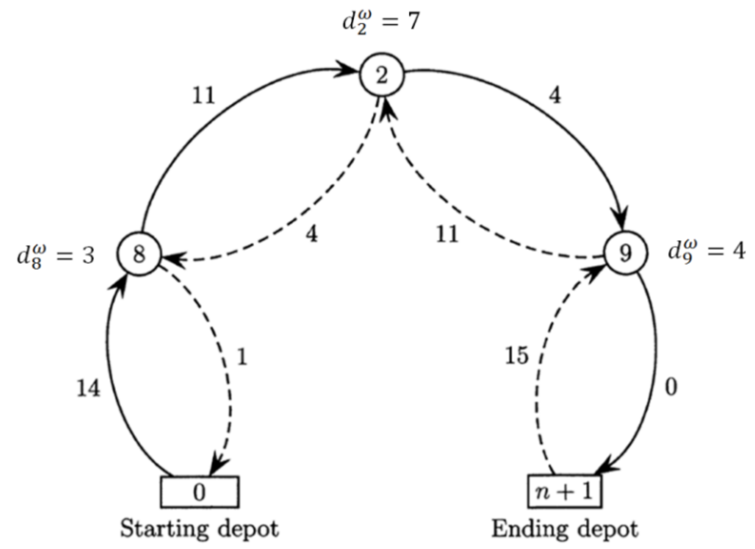

Figure 5: Example for (42)-(47), adapted from Baldacci et al. (2004)

Equation (42) ensures the total load of the vehicle is decreased by the demand of the visited client. The amount of empty space is controlled through (46), which states that if an arc is being used ($x_{ij}^{\omega} + x_{ji}^{\omega} = 1$), the sum of the total load and the empty space is equal to the vehicle capacity. Equation (43) sets the total starting load of all vehicles equal to the total demand of all clients. In a similar way, (45) sets the total capacity 'leaving the depot': the number of vehicles multiplied by the vehicle capacity. Constraints (44) ensures the amount of capacity that 'arrives' at the starting depot, is equal to the total capacity minus total demand.

Now that we have modeled the truck load and we have defined variables indicating the amount of empty space, we can restrict capacity by forcing the z-variables to be non-negative; Equation (47).

2.2.6 Comparison

There is one very important difference between modeling time and capacity: when we model time, we require to know if the time of service at each client is within the endogenous time windows. When we model capacity, we only need to assert that the total demand of the clients we visit does not exceed capacity. We do not need to know the total number of units already delivered.

This is also the reason our four methods to model time in Section 2.1 can be applied to capacity as well by introducing the q-variables, which keep track of the number of units already delivered. In general, every set of time constraints that sets the time of service at each client can be rewritten to a set of capacity constraints by using the q-variables.

The advantages and disadvantages of the four time models have already been discussed in Section 2.1.5. It is likely though, that when applied to capacity, they perform worse than specific capacity models. Take for example the 2-commodity flow formulation: there are no big-M terms necessary and the number of variables and constraints is reasonable. This can not be said for the MTZ-inequalities and the affine combination inequalities, which use big-M terms. The convex hull formulation does not require a big-M, but at the cost of a big increase in the number of variables and constraints. The only time model that seems to be able to compete with the 2-commodity flow formulation is the improved convex hull formulation. If the 2-commodity flow formulation is able to exploit some capacity specific property, however, it may very well be better than the improved convex hull formulation.

Overall, the capacity specific constraints thus seem more useful than the capacity constraints adapted from time constraints. This is also true for the generalized subtour elimination constraints, which have proven very effective (Baldacci et al. (2012), Spliet and Gabor (2014)). The obvious downside of using the generalized subtour elimination constraints is the exponential number of constraints. Often, the generalized subtour elimination constraints are added during the process of branch-and-cut, so only violated constraints are added.

2.3 Path inequalities

Another way to handle both time and capacity constraints, is by using path inequalities. These inequalities have first been proposed by Ascheuer et al. (2001) to solve the TSP with time windows, and have later been used by others to solve the VRPTW (see for example Kallehauge (2008)).

The idea is to add one constraint for each infeasible path, to make sure only feasible paths are chosen. This can be accomplished by using the following constraints:

$$\sum_{(k,l)\in E_p} x_{kl}^{\omega} \leq |E_p| - 1 \;\forall \text{ paths } P = (V_p, E_p) \subset G, P \text{ infeasible}, \omega \in \Omega \qquad (48)$$

An advantage of this method is that it is relatively easy to place additional constraints on the paths, as infeasible paths are simply disallowed. The disadvantage this brings is that the number of constraints is exponentially large, hence finding and adding cuts during branch-and-cut will be necessary. Another disadvantage is that infeasible path inequalities can be very weak in the current form, and would require problem specific strengthening to be really effective (Ascheuer et al., 2001).

3 Valid inequalities

In the previous section, we have discussed the various options to model time and capacity in the general flow formulation. When combined with a set of time constraints and a set of capacity constraints, the general flow formulation allows for solving the TWAVRP to optimality by applying a typical branch-and-bound algorithm. It is, however, generally very slow to solve VRP-like problems in this way, as the LP relaxations give weak lower bounds.

To overcome this problem we introduce various classes of valid inequalities, which can be used in a branch-and-cut algorithm to strengthen the general flow formulation. As the literature provides us with a large amount of different classes of valid inequalities that are applicable to the TWAVRP, we will make a selection of high potential classes and discuss those only. Also, novel inequalities, specifically for the TWAVRP, will be introduced.

3.1 Generalized subtour elimination constraints

The generalized subtour elimination constraints have already been mentioned as a way of modeling capacity in Section 2.2.4. As mentioned before, it is hard to model capacity using the generalized subtour elimination constraints, as there is an exponential number of constraints to be added. Say we model capacity by the generalized subtour elimination constraints, and these are the only capacity constraints we use. We are then at one point required to add *all* violated constraints, or the formulation becomes invalid. Exact separation, however, can be very time consuming.

We get the best of both worlds when we use a different method of modeling capacity, and only separate the generalized subtour elimination constraints by a heuristic. That is, the added inequalities do no longer guarantee that the vehicle capacity is respected, but do add strength to the already correct formulation.

Though it is possible to use heuristics to find violations of the generalized subtour elimination constraints (41), it is common in the literature to use heuristic separation on the easier variant in which λ_S^ω is replaced by $\lceil \left(\sum_{i \in S} d_i^\omega\right) / Q \rceil$ (Toth and Vigo, 2001). This yields:

$$\sum_{(i,j) \in (S, V \setminus S)} x_{ij}^\omega \geq \left\lceil \left(\sum_{i \in S} d_i^\omega \right) / Q \right\rceil \quad \forall S \subset V, \omega \in \Omega \tag{49}$$

The inequalities of Equation (49) are known as the *rounded capacity inequalities*. For notational convenience, we will refer to them as *capacity cuts* during the remainder of this thesis.

Note that the same inequalities have been used by Spliet and Gabor (2014) and have been found very effective for the TWAVRP. Furthermore, adding the capacity cuts diminished the effect of all other valid inequalities they tested.

3.2 One-way arc constraints

Consider a feasible solution to the TWAVRP, which contains two nodes i and j. In a single scenario, there are only three cases possible: j is visited after i, i is visited after j or i and j are not visited consecutively. In other words, all arcs can only be traveled in one direction.

We can turn this fact into the following set of valid inequalities, which we will call the one-way arc constraints:

$$x_{ij}^{\omega} + x_{ji}^{\omega} <= 1 \ \forall i \in V, \forall j \in V, \omega \in \Omega \tag{50}$$

The big advantage of the one-way arc constraints is that the number of inequalities is small enough to just add them all to the general formulation, without the need of separation.

3.3 Precedence inequalities

The previously discussed valid inequalities were applicable to each scenario separately, but we have not yet seen cuts that take multiple scenarios into account at once. To the best of my knowledge, such valid inequalities have not yet been proposed in the literature. In this section, I will present the precedence inequalities, a novel set of inequalities specifically for the TWAVRP.

First, we will need to define the following:

E_p	set of arcs of path p in G
$\mathscr{P}_{ij}$	set of all elementary paths p in G from $i \in V$, to $j \in V$
$\delta_{ij}(F, S)$	shortest distance from $i \in V$ to $j \in V$ using only arcs from set $F \in E$ and visiting all nodes in $S \subseteq V' \backslash \{i, j\}$

Now we present the main theorem necessary to define the precedence inequalities:

Theorem 1. *For given nodes $i, j \in V'$ ($i \neq j$), for any feasible solution to the TWAVRP in which both path $p \in \mathscr{P}_{ij}$ is used in scenario $\omega \in \Omega$ and path $q \in \mathscr{P}_{ji}$ is used in scenario $\omega' \in \Omega$ the following holds:*

$$\sum_{(k,l) \in E_p} t_{kl} + \sum_{(k,l) \in E_q} t_{kl} \leq w_i + w_j \tag{51}$$

Proof. A necessary condition for both $p \in \mathscr{P}_{ij}$ to be chosen in scenario $\omega \in \Omega$ and $q \in \mathscr{P}_{ji}$ to be chosen in scenario $\omega' \in \Omega$ is the following:

$\exists \ y_i \in [s_i, e_i - w_i], y_j \in [s_j, e_j - w_j]$ such that (52) and (53) hold.

$$y_i + \sum_{(k,l) \in E_p} t_{kl} \leq y_j + w_j \tag{52}$$

$$y_j + \sum_{(k,l) \in E_q} t_{kl} \leq y_i + w_i \tag{53}$$

The interpretation is clear: the left hand side is the earliest arrival time at the destination node, as it is the sum of the earliest departure time and the travel times over the path. The right hand side is the latest possible arrival time at the destination according to the time windows. If both paths p and q are being used, it is required for both paths that it is possible to arrive before the end of the time window.

Adding (52) to (53) results in

$$y_i + y_j + \sum_{(k,l) \in E_p} t_{kl} + \sum_{(k,l) \in E_q} t_{kl} \leq y_i + y_j + w_i + w_j \tag{54}$$

which directly implies (51) as the y_i and y_j cancel. It follows that (51) is a necessary condition for using both path p in scenario ω and path q in scenario ω'. $\square$

Theorem 1 tells us what the consequences for the other scenarios are when we choose a path in one scenario. For example: say there is a single truck that visits first i and later j somewhere on the same day. Between i and j, it has driven a total of 5 hours. Furthermore, say both i and j have an endogenous time window width of 3 hours. We can then deduce that if we want to plan any path from j to i in another scenario, we can only use paths with at most $3 + 3 - 5 = 1$ hour of travel time between j and i.

Now that we have established which paths exclude each other, we will develop a simple test to see whether a set of nodes contains a path visiting all nodes of the set. This test will later be used to define valid inequalities.

To this end, we state Theorem 2 and Theorem 3 without proof. The correctness of these theorems follows directly from the correctness of the subtour elimination constraints for the traveling salesman problem (see for example Padberg and Rinaldi (1991)).

Theorem 2. *For any feasible solution to the TWAVRP the following holds:*

$$\sum_{k \in S} \sum_{l \in S} x_{kl}^{\omega} \leq |S| - 1 \ \forall \omega \in \Omega, S \subseteq V', S \neq \emptyset \tag{55}$$

Theorem 3. *For a feasible TWAVRP solution, a non-empty set of customers $S \subseteq V'$ is consecutively served by a single truck in scenario $\omega \in \Omega$ if and only if:*

$$\sum_{k \in S} \sum_{l \in S} x_{kl}^{\omega} = |S| - 1 \tag{56}$$

Figure 6 gives an example of a situation where (56) holds.

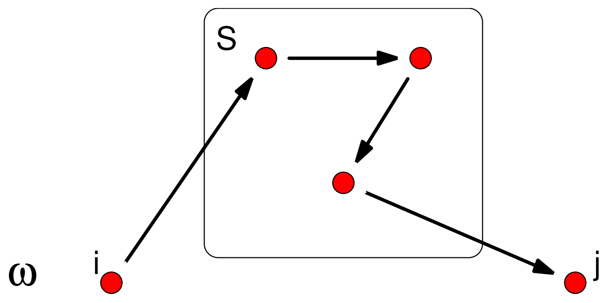

Figure 6: Example where (56) holds. As $|S| = 3$ the number of arcs contained within S is at most $|S| - 1 = 2$

Theorem 4. *For a feasible TWAVRP solution, a set of customers $S \subseteq V'$ and two nodes $i, j \in V' \backslash S$ $(i \neq j)$ a single truck visits first i, then all customers in S and then j consecutively in scenario $\omega \in \Omega$ if and only if:*

$$\sum_{l \in S} x^{\omega}_{il} + \sum_{k \in S} \sum_{l \in S} x^{\omega}_{kl} + \sum_{k \in S} x^{\omega}_{kj} + x^{\omega}_{ij} = |S| + 1 \tag{57}$$

Proof. $\Longrightarrow$: the described path requires both a single arc from i to a node in S and a single arc from a node in S to j in case $S \neq \emptyset$. These arcs force x^{ω}_{ij} to be zero due to the flow constraints. Combining this with Theorem 3 proves the implication. If $S = \emptyset$ we have $x^{\omega}_{ij} = 1$ and the implication still holds.

$\Longleftarrow$: assume $S \neq \emptyset$. By Theorem 2, $\sum_{k \in S} \sum_{l \in S} x^{\omega}_{kl} \leq |S| - 1$. Due to the flow constraints $\sum_{l \in S} x^{\omega}_{il} + \sum_{k \in S} x^{\omega}_{kj} + x^{\omega}_{ij} \leq 2$. It thus follows from Theorem 2 that both inequalities hold with equality. This implies the customers in S are all connected by a single path (Theorem 3). Because of the flow constraints, $x^{\omega}_{ij} = 0$ and hence $\sum_{l \in S} x^{\omega}_{il} = 1$ and $\sum_{k \in S} x^{\omega}_{kj} = 1$. Again using the flow constraints, it is necessary that there is a path visiting i, all nodes in S and then j, in that order. Note that $S = \emptyset$ implies $x^{\omega}_{ij} = 1$ so the theorem still holds. □

Theorem 4 gives us a simple way to test whether there is a path visiting i, then visiting a subset of customers, and then visiting j. Combining this theorem with Theorem 1 allows us to find invalid combinations of a path from i to j in one scenario, and a path from j to i in another scenario. Let us use the notation $(A : B)$ to indicate the set of arcs in E that start in A and end in B, for A and B being nodes or sets of nodes. We then get the following theorem:

Theorem 5. *For given scenarios $\omega, \omega' \in \Omega$ $(\omega \neq \omega')$, given nodes $i, j \in V'$ $(i \neq j)$, given node sets $S \subseteq V' \backslash \{i, j\}$, $S' \subseteq V' \backslash \{i, j\}$ and given arc sets $F \subseteq (i : S) \cup (S : S) \cup (S : j) \cup (i : j)$, $F' \subseteq (j : S') \cup (S' : S') \cup (S' : i) \cup (j : i)$ such that $\delta_{ij}(F, S) + \delta_{ji}(F', S') > w_i + w_j$, the following are valid inequalities:*

$$\sum_{(k,l) \in F} x^{\omega}_{kl} + \sum_{(k,l) \in F'} x^{\omega'}_{kl} \leq |S| + |S'| + 1 \tag{58}$$

Proof. A direct application of Theorem 1 and Theorem 4 shows that Theorem 1 is violated if and only if the left hand side of (58) is equal to $|S| + |S'| + 2$. By integrality of the x-variables, the theorem follows. $\square$

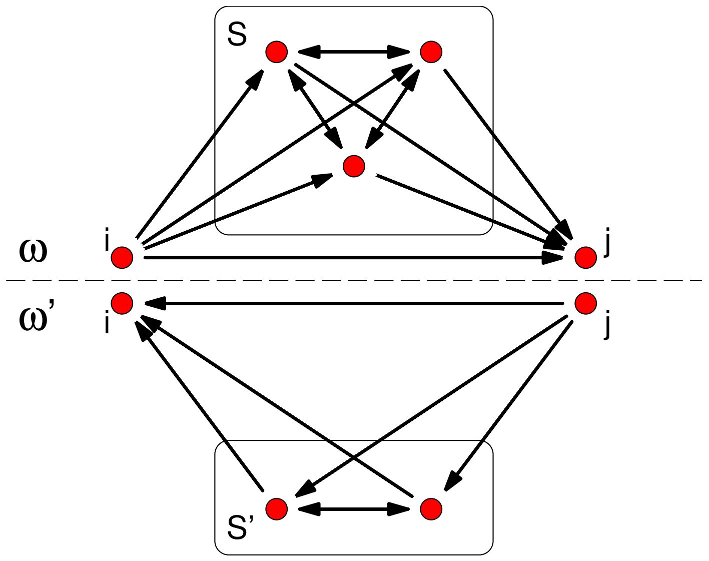

Figure 7: Example showing all arcs in F and F', for $|S| = 3$ and $|S'| = 2$. If there are two arcs between two points, this is depicted by a double-headed arrow. Any path over F and F' visiting all clients in S and S' requires exactly $|S| + |S'| + 2 = 7$ arcs.

Figure 7 gives an example of the sets F, F', S and S'.

It is possible to state a slightly stronger result by redefining $\delta_{ij}(F, S)$ to be the minimum distance to visit i, all clients in S and then j using only arcs of F, but only using paths that can be feasible when considering the exogenous time windows. In practice, the exogenous time windows are generally very large. Doing so is thus unlikely to add much value. For this reason, we use this likely less complicated inequality.

We have thus found an exponential number of valid inequalities that can be added. Again, it is not efficient to add all these valid inequalities from the start. If we can (heuristicly) separate the violated inequalities fast, it may prove beneficial to add some during branch-and-cut.

3.4 Sub-optimal path inequalities

Besides the precedence inequalities, we will introduce another class of inequalities, specifically for the TWAVRP.

Consider an optimal TWAVRP solution in which a set of clients $S \subseteq V'$ is being served by a single vehicle in scenario $\omega \in \Omega$, and no other clients are served by the same vehicle. Clearly, the vehicle uses the cheapest path visiting all clients in S and adhering to the time window constraints. If it would not be the cheapest path, the solution would not be optimal.

As the vehicle uses the cheapest path, there can not be any path which is strictly cheaper. If the same situation occurs in another scenario, it is optimal to travel the exact same path. This is always possible, because the time window constraints do not change between scenarios. Hence we can conclude the following: if two vehicles serve the same set of clients, and no other clients, in different scenarios, there exists an optimal solution in which both vehicles drive the same route. We can capture this by the following theorem:

Theorem 6. *For given scenarios $\omega, \omega' \in \Omega$ $(\omega \neq \omega')$, given paths $p, q \in \mathscr{P}_{0,n+1}$ such that $V_p = V_q$ and $E_p \neq E_q$, the following are valid inequalities:*

$$\sum_{(k,l)\in E_p} x^{\omega}_{kl} + \sum_{(k,l)\in E_q} x^{\omega'}_{kl} \leq |E_p| + |E_q| - 1 \tag{59}$$

Proof. Paths p and q visit the same clients $(V_p = V_q)$, but in a different order $(E_p \neq E_q)$. As explained above, there always exists an optimal solution for which in each scenario the same path is taken if all the same clients are being served by a single vehicle (and no other clients). We can thus state only path p or path q can be used. Because p and q are both used if and only if the left hand side equals $|E_p| + |E_q|$ (see Section 2.3), the inequality follows. $\square$

Note that the sub-optimal path inequalities can cut off feasible and even optimal solutions. In the literature, such inequalities are often referred to as *optimality cuts.* Feasible solutions are cut off when two routes visiting the same set of clients in a different order have the same costs. The sub-optimal path inequalities do not alter the optimal objective value.

3.4.1 Relation to precedence inequalities

The precedence inequalities and the sub-optimal path inequalities are disjunct sets, as the precedence inequalities never involve the depot while the sub-optimal path inequalities always involve the depot. It is possible, however, for a precedence inequality to dominate a sub-optimal path inequality.

For example, the precedence inequality

$$x^{\omega}_{ij} + x^{\omega'}_{ji} <= 1 \tag{60}$$

dominates the sub-optimal path inequality

$$x^{\omega}_{0i} + x^{\omega}_{ij} + x^{\omega}_{j,n+1} + x^{\omega'}_{0j} + x^{\omega'}_{ji} + x^{\omega'}_{i,n+1} <= 5 \tag{61}$$

There do exist sub-optimal path inequalities that are not dominated by precedence inequalities. For example, if $t_{ij} + t_{ji} \leq w_i + w_j$, inequality (60) becomes invalid, while (61) can still be used.

4 Cut separation

In Section 3, we have discussed multiple classes of valid inequalities. Though in theory we could add all valid inequalities to the general flow formulation from the beginning, this is often not possible in practice as the number of valid inequalities can be very big. To solve this problem, we will add the valid inequalities only when they are violated. This way, valid inequalities that have no effect, do not have to be added.

We will refer to the process of finding violated valid inequalities as *cut separation.* In this section we will discuss the separation algorithms to separate the classes of valid inequalities as discussed in the previous section.

4.1 Capacity cuts

The problem of separating the generalized subtour elimination constraints is known to be strongly NP-hard (Lysgaard et al., 2004). Lysgaard et al. do not state whether this is also true for the rounded capacity cuts. Regardless of its classification, the separation problem is difficult, and the use of heuristics is justified (Toth and Vigo, 2001).

To separate the capacity cuts, we use the CVRPSEP package of Lysgaard (2003), which uses a variety of heuristics to find cut violations. The details can be found in the paper of Lysgaard et al. (2004).

As the package of Lysgaard requires an upper bound on the number of capacity cuts to be separated per iteration, we set the maximum to 1000, which after some preliminary testing seems to be non restrictive.

We found that the package of Lysgaard can end up in an infinite loop when, for any $\omega \in \Omega$

$$\sum_{j \in V'} x_{0j}^{\omega} = \sum_{i \in V'} x_{i,n+1}^{\omega} = 0 \tag{62}$$

To overcome this problem, we add a similar constraint to make sure the number of vehicles leaving the depot is nonzero. That is, we add

$$\sum_{j \in V'} x_{0j}^{\omega} \geq \left\lceil \frac{\sum_{i \in V'} d_i^{\omega}}{Q} \right\rceil \tag{63}$$

4.2 Precedence inequalities

The complexity of the separation problem of the precedence inequalities is not known, as the precedence inequalities have only been introduced in this thesis. We will first discuss exact separation of the precedence inequalities, and discuss two heuristics afterwards.

4.2.1 Exact separation

To find a violated precedence inequality with certainty (if one exists) we can use another MIP, in which the x-values are fixed parameters corresponding to the (fractional) values of the LP relaxation. Whether exact separation speeds up the optimization is questionable, as it may be rather time consuming to solve the separation problem. It can, however, be a good starting point for a heuristic. Exact separation is also useful to analyze how the precedence inequalities affect the lowerbounds and the size of the branch-and-cut tree.

The MIP formulation requires two scenarios to be chosen in advance, ω and ω' ($\omega, \omega' \in \Omega$, $\omega \neq \omega$). We define the following binary variables:

$$
\begin{array}{ll}
a_i & \begin{cases} 1 & \text{if } i \in V' \text{ represents the `starting node' in scenario } \omega \\ 0 & \text{otherwise} \end{cases} \\
b_j & \begin{cases} 1 & \text{if } j \in V' \text{ represents the `ending node' in scenario } \omega \\ 0 & \text{otherwise} \end{cases} \\
s_i^\omega & \begin{cases} 1 & \text{if node } i \in S \text{ or } a_i = 1 \text{ or } b_i = 1 \\ 0 & \text{otherwise} \end{cases} \\
s_i^{\omega'} & \begin{cases} 1 & \text{if node } i \in S' \text{ or } a_i = 1 \text{ or } b_i = 1 \\ 0 & \text{otherwise} \end{cases} \\
f_{ij}^\omega & \begin{cases} 1 & \text{if arc } (i,j) \in (V', V') \text{ is selected in scenario } \omega \\ 0 & \text{otherwise} \end{cases} \\
f_{ij}^{\omega'} & \begin{cases} 1 & \text{if arc } (i,j) \in (V', V') \text{ is selected in scenario } \omega' \\ 0 & \text{otherwise} \end{cases}
\end{array}
$$

The formulation is then given by:

Precedence inequalities: exact separation

$$\max \sum_{i \in V'} \sum_{j \in V'} (f_{ij}^\omega x_{ij}^\omega + f_{ij}^{\omega'} x_{ij}^{\omega'}) - \sum_{i \in V'} (s_i^\omega + s_i^{\omega'}) + 3 \tag{64}$$

$$\sum_{i \in V'} a_i = 1 \tag{65}$$

$$\sum_{j \in V'} b_j = 1 \tag{66}$$

$$a_i + b_i \leq 1 \;\forall i \in V' \tag{67}$$

$$s_i^\omega, s_i^{\omega'} \geq a_i, b_i \;\forall i \in V' \tag{68}$$

$$f_{ij}^\omega \leq s_i^\omega, s_j^\omega \;\forall i,j \in V' \tag{69}$$

$$f_{ij}^{\omega'} \leq s_i^{\omega'}, s_j^{\omega'} \;\forall i,j \in V' \tag{70}$$

$$f_{ji}^\omega \leq 1 - a_i \;\forall i,j \in V' \tag{71}$$

$$f_{ji}^\omega \leq 1 - b_j \;\forall i,j \in V' \tag{72}$$

$$f_{ij}^{\omega'} \leq 1 - a_i \; \forall i,j \in V' \tag{73}$$

$$f_{ij}^{\omega'} \leq 1 - b_j \; \forall i,j \in V' \tag{74}$$

$$\sum_{(i,j)\in E_p} f_{ij}^{\omega} + \sum_{(i,j)\in E_q} f_{ij}^{\omega'} \leq \sum_{i\in V'} (s_i^{\omega} + s_i^{\omega'}) - 3$$

$$\forall k,l \in V', k \neq l, p \in \mathscr{P}_{kl}, q \in \mathscr{P}_{lk}, \sum_{(i,j)\in E_p} t_{ij} + \sum_{(i,j)\in E_q} t_{ij} \leq w_k + w_l \tag{75}$$

$$a_i, b_i, s_i^{\omega}, s_i^{\omega'} \in \mathbb{B} \; \forall i \in V' \tag{76}$$

$$f_{ij}^{\omega}, f_{ij}^{\omega'} \in \mathbb{B} \; \forall i,j \in V' \tag{77}$$

This formulation is best explained by comparing it to Theorem 5. Note that the s^{ω}-variables are equal to one if and only if the corresponding node is an element of $\{i\} \cup S \cup \{j\}$ in Theorem 5, and hence $\sum_{i\in V'} s_i^{\omega} = |S| + 2$. The f^{ω} variables are equal to one if and only if the corresponding arc is a member of set F in Theorem 5. The a-variables indicate which node is used as the 'starting node' (i in Theorem 5) and the b-variables indicate the 'ending node' (j in Theorem 5).

The objective of the separation problem (64) is to maximize the amount the precedence inequality is violated. That is, the objective is equal to:

$$\sum_{(k,l)\in F} x_{kl}^{\omega} + \sum_{(k,l)\in F'} x_{kl}^{\omega'} - (|S| + |S'| + 1) \tag{78}$$

Constraints (65)-(67) ensure exactly one 'starting node' and one 'ending node' are chosen, and they are not the same. These nodes are then added to the set S and S' (in terms of Theorem 5) by constraint (68). To ensure we only select arcs as specified in Theorem 5, we add constraints (69)-(74). The first two constraints force all chosen arcs to be between the 'starting node', the 'ending node' and the set S or S'. This is not restrictive enough, as for example arcs from S to i are not allowed in Theorem 5. Constraints (71)-(74) deal with these restrictions.

Constraint (75) handles the final premise of Theorem 5, which states that the shortest path visiting all nodes in S and S' must be longer than the sum of the endogenous time window widths of the starting node and the ending node. Such a path visiting all nodes consists of $|S|+|S'|+2$ edges. The right hand side of (75) restricts the number of edges for such paths to $|S|+|S'|+1$. This implies that when a path does not visit all nodes in S and S', (75) is not restrictive. It is important to mention that (75) introduces a large number of constraints, so it may be beneficial to only add them if they are violated. This can be achieved by checking for violated constraints at each integer solution, and adding cuts if necessary. A violated constraint can be found by a simple depth-first search. That is, we follow an arbitrary path for which the f-variables of all arcs are equal to 1, and backtrack if we do not find a violation of (75).

Thus, to perform exact separation, we solve the MIP given by (64)-(77) for each combination of scenarios ω and ω' ($\omega \neq \omega'$). If the objective (64) is strictly positive, we have found a violated precedence inequality.

Combined with the fact that we need another separation algorithm to add the constraints (75), renders it unlikely that exact separation will be fast. Furthermore, it turns out that under certain assumptions, exact separation is NP-hard (see Appendix C for a proof). This justifies exploring heuristics to solve the separation problem of the precedence inequalities.

4.2.2 Separation by paths

Exact separation of the precedence inequalities may be too time consuming to be useful in a branch-and-cut framework. To this end, we also consider a subset of the precedence inequalities, which are easier to separate. We first introduce the following theorem:

Theorem 7. *For any feasible solution to the TWAVRP:*

$$\sum_{(k,l)\in E_p} t_{kl} + \sum_{(k,l)\in E_q} t_{kl} > w_i + w_j \implies \sum_{(k,l)\in E_p} x_{kl}^{\omega} + \sum_{(k,l)\in E_q} x_{kl}^{\omega'} \leq |E_p| + |E_q| - 1$$

$$\forall (i,j) \in E, p \in \mathscr{P}_{ij}, q \in \mathscr{P}_{ji}, \omega \in \Omega, \omega' \in \Omega \tag{79}$$

Proof. Direct application of Theorem 5 with $F = E_p$ and $F' = E_q$. Note that $|S| = |E_p| - 1$ and $|S'| = |E_q| - 1$, and hence $|S| + |S'| + 1 = |E_p| + |E_q| - 1$. □

Theorem 7 defines valid inequalities for paths only, instead of for arbitrary sets of nodes and arcs. Recall that Theorem 5 requires $\delta_{ij}(F,S) + \delta_{ji}(F',S') > w_i + w_j$ to hold. That is, the shortest path visiting all nodes has to have a minimum length. By only considering paths, checking whether this condition is met becomes trivial. We will now show that this subset of precedence inequalities can be separated in polynomial time.

Formally, we need to find some combination of nodes $i, j \in V'$, paths $p \in \mathscr{P}_{ij}, q \in \mathscr{P}_{ji}$ and scenarios $\omega, \omega' \in \Omega$ $(\omega \neq \omega')$ such that the following two conditions are met:

$$\sum_{(k,l)\in E_p} t_{kl} + \sum_{(k,l)\in E_q} t_{kl} > w_i + w_j \tag{80}$$

$$\sum_{(k,l)\in E_p} x_{kl}^{\omega} + \sum_{(k,l)\in E_q} x_{kl}^{\omega'} > |E_p| + |E_q| - 1 \tag{81}$$

Finding all such paths can be done in polynomial time. We will first prove three necessary lemmas, after which we will show how to solve the separation problem in $O(|\Omega|^2 n^6)$ time.

Lemma 1. *All solutions to the separation problem adhere to the following two equations:*

$$\sum_{(k,l)\in E_p} x_{kl}^{\omega} > |E_p| - 1 \tag{82}$$

$$\sum_{(k,l)\in E_q} x_{kl}^{\omega'} > |E_q| - 1 \tag{83}$$

Proof. The x variables are constrained to be between 0 and 1. This implies that $\sum_{(k,l)\in E_p} x_{kl}^{\omega} \leq |E_p|$. For (81) to hold, it is thus necessary that (82) holds. Analogously it can be shown (83) is necessary. □

Lemma 2. *Path p in graph G can contain at most one arc (k,l) for which $x_{kl}^{w} \leq \frac{1}{2}$ and path q can contain at most one arc (k',l') for which $x_{k'l'}^{\omega'} \leq \frac{1}{2}$.*

Proof. Say p has $m \geq 2$ arcs for which $x_{kl}^{\omega} \leq \frac{1}{2}$. This implies $\sum_{(k,l)\in E_p} x_{kl}^{\omega} \leq \frac{1}{2}m + \sum_{(k,l)\in E_p, x_{kl}^{\omega} > \frac{1}{2}} x_{kl}^{\omega} \leq \frac{1}{2}m + |E_p| - m = |E_p| - \frac{1}{2}m \leq |E_p| - 1$. Hence the conditions in Lemma 1 are not satisfied. It follows p has at most one arc for which $x_{kl}^{\omega} \leq \frac{1}{2}$. The proof for path q is analogous. □

Lemma 3. *For each scenario $\omega \in \Omega$, node $k \in V'$ has at most one outgoing arc (k,l) such that $x_{k,l}^{\omega} > \frac{1}{2}$ and at most one incoming arc (m,k) such that $x_{m,k}^{\omega} > \frac{1}{2}$.*

Proof. The sum of the x variables on the outgoing arcs is restricted to 1. As the x variables are positive, this restriction can only be met when there is at most one outgoing arc $x_{k,l}^{\omega} > \frac{1}{2}$. The proof for the incoming arcs is analogous. □

Theorem 8. *The separation problem can be solved in $O(|\Omega|^2 n^6)$ time.*

Proof. To solve the separation problem, we will generate a list of candidate paths from i to j which meet the necessary conditions given by the lemmas. If we do the same for all candidate paths from j to i, we can check for all combinations of the candidates if (80) and (81) are met.

To generate a list of candidates, we first use Lemma 2, which states that for scenario $\omega \in \Omega$ a candidate uses at most one arc for which $x_{kl}^{\omega} \leq \frac{1}{2}$. Starting at i, the path thus first uses a (possibly zero) number of arcs for which $x_{kl}^{\omega} > \frac{1}{2}$, followed by zero or one arcs for which $x_{kl}^{\omega} \leq \frac{1}{2}$. After that, we visit another (possibly zero) number of arcs for which $x_{kl}^{\omega} > \frac{1}{2}$ before we reach j.

From Lemma 3 we know that at each node, there can be at most one incoming arc and one outgoing arc for which $x_{kl}^{\omega} > \frac{1}{2}$. This implies there is at most one elementary path leaving i for which all arcs have an x value larger than $\frac{1}{2}$. Analogously there is at most one elementary path entering j for which all x values are larger than $\frac{1}{2}$.

All candidate paths from i to j can thus be constructed by starting in i, following the arcs with x values larger than $\frac{1}{2}$ up to a certain point after which an arc with x value less or equal to $\frac{1}{2}$ is taken to arrive at the path of arcs with x values larger than $\frac{1}{2}$ that arrives at j, which is followed until we reach j.

For each scenario, we can do this in $O(n^2)$ ways. We thus find $O(|\Omega|n^2)$ candidates from i to j. Checking all combinations of the candidates from i to j and the candidates from j to i then involves $O((|\Omega|n^2)^2)$ time. As we repeat the procedure for all combinations of two nodes, the total time complexity is $O(|\Omega|^2 n^6)$. □

4.2.3 Separation by DAGs

In the previous section, we have separated the precedence inequalities by simple paths. That is, we only separated precedence inequalities for which, using the notation of Theorem 5, arc sets F and F' belonged to simple paths. The benefit of doing so, is that it is immediately clear whether combinations of paths are long enough to form a precedence inequality.

Simple paths are not the only subset for which this is true: we can also separate by directed acyclic graphs (DAGs). That is, we restrict F and F' to belong to DAGs. DAGs have the property that there is at most one path visiting all nodes in the subgraph, and hence, the minimum time to visit all nodes is easy to calculate: we simply follow the unique path through the DAG that visits all nodes, and keep track of the necessary time to do so.

Figures 8 and 9 show the difference between a precedence inequality made from paths and one made from DAGs, respectively. It can be seen that the precedence inequality constructed from the DAGs contains more arcs, and hence is a stronger inequality. The downside of separating by DAGs, is that the separation itself becomes more difficult.

To construct precedence inequalities from DAGs, we first produce a list of candidate DAGs, just like we made a list of candidate paths in Section 4.2.2. We can generate this list by using Algorithm 1, which in turn makes use of Algorithm 2.

Algorithm 1 FIND_CANDIDATE_DAGS

```
1: Initialize empty list of candidates
2: for all $\omega \in \Omega$ do
3:     for all $StartNode \in V'$ do
4:         $V_p \leftarrow \{StartNode\}$
5:         $E_p \leftarrow \emptyset$
6:         EXTEND($\omega$, $V_p$, $E_p$, 0, 0, $StartNode$)
7:     end for
8: end for
9: return candidate list
```

Algorithm 1 uses Algorithm 2 to generate candidate DAGs, for each starting node in each scenario. This is shown by the call to EXTEND on line 6. Algorithm 2 then generates all candidate DAGs starting from this node.

Algorithm 2 is a recursive algorithm that extends the current DAG given by (V_p, E_p) in all possible ways. While extending, the algorithm keeps track of the sum of all x-variables contained in E_p ($SumX$) and the total time necessary to visit all nodes by following the DAG ($SumTime$). As soon as a candidate DAG is found, it is added to the candidate list.

Lines 1-2 ensure the path is only further extended when the necessary condition $\sum_{(k,l)\in E_p} x^{\omega}_{kl} > |E_p| - 1$ is met. This necessary condition follows from Lemma 1 and the fact that in each iteration of EXTEND, $SumX$ increases by

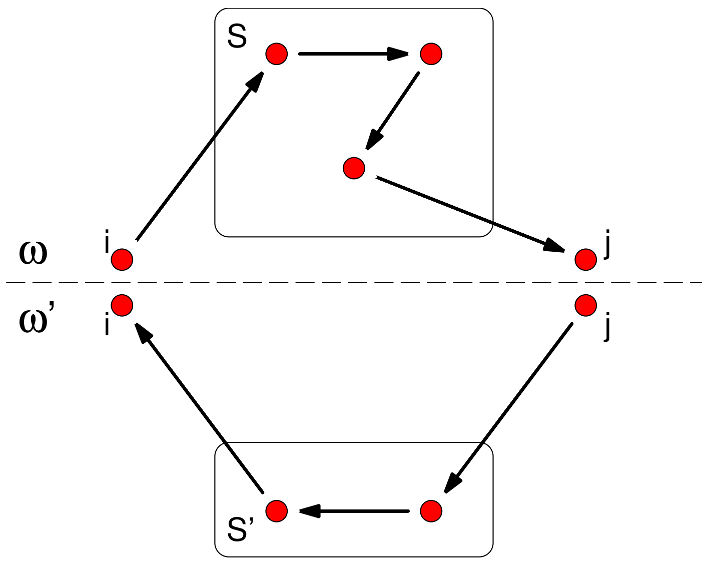

Figure 8: Precedence inequality given by two paths, one in scenario ω and one in scenario ω'

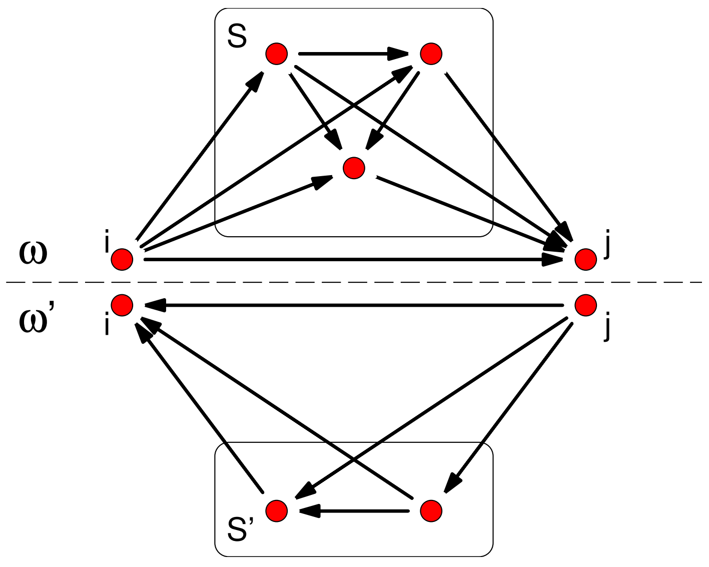

Figure 9: Precedence inequality given by two DAGs, one in scenario ω and one in scenario ω'

Algorithm 2 EXTEND($Scenario$, V_p, E_p, $SumX$, $SumTime$, $CurrentNode$)

```
1: if $SumX \leq |E_p| - 1$ then
2:     Do not continue recursion
3: else
4:     if $|E_p| \geq 1$ then
5:         Add path candidate ($V_p$,$E_p$,$SumX$,$SumTime$) to candidate list
6:     end if
7:     for all $NextNode \in V' \backslash V_p$ do
8:         $SumTime \longleftarrow SumTime + t_{CurrentNode,NextNode}$
9:         for all $PreviousNode \in V_p$ do
10:            $E_p \longleftarrow E_p \cup \{(PreviousNode, NextNode)\}$
11:            $SumX \longleftarrow SumX + x^{Scenario}_{PreviousNode,NextNode}$
12:        end for
13:        $V_p \longleftarrow V_p \cup \{NextNode\}$
14:        EXTEND($Scenario$, $V_p$, $E_p$, $SumX$, $SumTime$, $CurrentNode$)
15:    end for
16: end if
```

at most 1 (due to constraint (3) in the general flow formulation). If the neccesary condition is met, and the generated DAG is not empty, it is added to the candidate list. This is stated in lines 4-6. Lines 7-15 extend the DAG by a single node (line 13) and adds all directed arcs from the previous DAG to this new node (lines 9-12). The total travel time and the sum of the x-variables on the DAG are updated accordingly (lines 8 and 11 respectively). Finally we recursively call for further extension of the DAG on line 14.

4.2.4 Additional strategies

We have discussed three different methods for separating precedence inequalities: the exact method, separation by paths and separation by DAGs.

There are, however, some additional strategies that can be utilized. For example, one can use separation by paths to find candidate paths, and then turn each path into a DAG before constructing the precedence inequalities. This yields stronger valid inequalities than using separation by paths alone.

Do note that by using separation by paths, less violations will be found then when using separation by DAGs. Figure 10 shows an example in which the DAG violates the precedence inequality, but the associated path does not. In this case, the path precedence inequality corresponds to the path $i \rightarrow k \rightarrow l \rightarrow j$ in scenario ω and path $j \rightarrow i$ in scenario ω', yielding the non-violated inequality:

$$x^{\omega}_{ik} + x^{\omega}_{kl} + x^{\omega}_{lj} + x^{\omega'}_{ji} \leq 3 \tag{84}$$

The inequality corresponding to the shown DAGs, however, is violated:

$$x^{\omega}_{ik} + x^{\omega}_{il} + x^{\omega}_{kl} + x^{\omega}_{kj} + x^{\omega}_{lj} + x^{\omega'}_{ji} \leq 3 \tag{85}$$

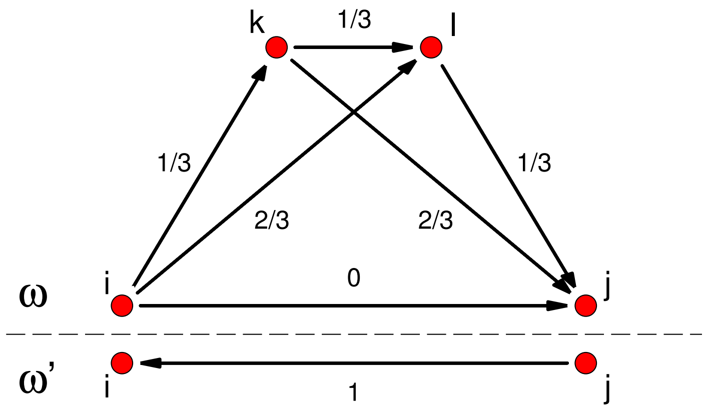

Figure 10: Precedence inequality given by two DAGs, one in scenario ω and one in scenario ω'. The values on the arcs correspond to the values of the x-variables.

Another strategy could be to add even more arcs. Consider a candidate path or DAG from i to j, visiting all nodes in $S \subseteq V' \backslash \{i, j\}$. We can then also turn the subgraph implied by S into a complete graph. That is, we add all arcs with both nodes in S to the candidate. Note that in contrast to transforming a path into a DAG, adding a complete subgraph *does* alter the minimum time required to visit all nodes only using the chosen arcs. To be able to use the candidate to form precedence inequalities, we use a lowerbound on this minimum time. An easy to calculate lower bound is given in Theorem 9.

Theorem 9. *Consider the problem of finding the minimum time to visit i, then all nodes of $S \subseteq V' \backslash \{i, j\}$ and then j consecutively. A lowerbound on the minimum time is given by:*

$$\min_{k \in S} \{t_{ik}\} + MST(S) + \min_{k \in S} \{t_{kj}\} \tag{86}$$

In which $MST(S)$ depicts the weight of the minimum weight spanning tree of the complete graph implied by S, using the time distances as weights.

Proof. To visit i, all nodes of S then and j, it is necessary that all nodes in S are (indirectly) connected, and that both i and j are connected to one of the nodes in S. The best way to fulfill this necessary condition, is to calculate the MST of S and connect i and j in the best possible way.

As we meet one of the necessary conditions in the best possible way, we have determined a lower bound on the actual value. □

To summarize, if we find candidate paths, we can always turn them into a candidate DAGs to increase their strength. When we find candidate paths or DAGs, we can turn them into candidates containing a complete subgraph. If we do so, we do need to calculate a lower bound on the minimum time to visit all nodes, for example by using (86). This allows for stronger precedence inequalities to be found. If no precedence inequality containing a complete subgraph can be constructed, due to the calculated lower bound on the minimum time to visit all nodes, we can still use the original path or DAG precedence inequality.

4.3 Sub-optimal path inequalities

The sub-optimal path inequalities can be separated in way analogous to the separation by paths for precedence inequalities (Section 4.2.2), and can also be separated in polynomial time.

First, we create partial candidate paths, by creating candidates without the depot. We can do so in the same way as we did in Section 4.2.2. That is, we construct all partial candidate paths from i to j by starting in i, following the arcs with x values larger than $\frac{1}{2}$ up to a certain point after which an arc with x value less or equal to $\frac{1}{2}$ is taken to arrive at the path of arcs with x values larger than $\frac{1}{2}$ that arrives at j, which is followed until we reach j. This way, only partial candidates that meet the necessary condition stated in Lemma 1 are found.

Then, we add the depot nodes at the beginning and at the end of the found partial candidates. The candidate paths for the sub-optimal path inequalities are given by all paths that still meet the necessary condition stated in Lemma 1.

Then, all combinations of candidates are checked to see if they form a sub-optimal path inequality. That is, the premises of Theorem 6 are met. This procedure finds all violated sub-optimal path inequalities in polynomial time.

5 Branch-and-cut framework

In the previous sections, we have prepared all ingredients to build a successful branch-and-cut algorithm. In Section 2, we have discussed the general flow formulation, the basis of our algorithm. In Section 3 we introduced valid inequalities to strengthen the formulation and in Section 4 we discussed the separation of these valid inequalities.

In this section, we will define our branch-and-cut framework. We specify five things: our branching strategy, our method of determining lower bounds for each node of the tree, our method of determining upper bounds, which cuts to apply and our stopping criteria. As a basis, we use the well-known commercial solver CPLEX, version 12.5. CPLEX provides default settings for all five choices.

We will use the default settings for determining the upper- and lower bounds. By default, upper bounds are found by applying a variety of heuristics, and of course, when the solution is integral. Lowerbounds are found by solving the LP relaxation of the problem with the dual simplex method and adding valid inequalities.

We will not use the default dynamic branching algorithm, but the build in ‘traditional branch-and-cut’, which is closer to standard branch-and-cut.

CPLEX provides a large variety of cuts, which are all enabled by default. We disable all build in valid inequalities, so we can exclusively test the effect of the valid inequalities discussed in this thesis.

The default stopping criterion is to stop when the relative gap between upper- and lower bound reaches 0.0001. To be consistent with the tests of Spliet and Gabor (2014), we stop only when the relative gap becomes 0.000001. Also, we stop after one hour of runtime.

Our own valid inequalities will be generated in a so-called *user cut callback*. The code in this callback is called each time that the LP relaxation has been solved, or re-solved after adding valid inequalities. Pseudocode for determining the lower bounds is given by Algorithm 3.

See Appendix D for an overview of all parameter settings.

Algorithm 3 Pseudocode lower bounds branch-and-cut algorithm

```
Initialize problem
Add one-way arc constraints to general flow formulation
Set the root node to be the current node
while relative gap $> 0.00001$ and runtime $< 3600$ seconds do
    Solve LP relaxation current node
    while Violated valid inequalities may be found do
        Add violated capacity cuts
        Add violated precedence inequalities
        Add violated sub-optimal path inequalities
        Solve LP relaxation current node
    end while
    Determine new current node (CPLEX)
end while
```

6 Experiments

In this section, we describe the experiments. First we describe our test setting. Then we decide on the scope of the experiments, as there are too many combinations of formulations and valid inequalities to try them all. In Section 6.3 we will describe the experiments within this scope.

6.1 Test setting

To allow for a fair comparison with their results, Spliet and Gabor (2014) provided both the 40 test-instances they have used themselves, plus an additional 20 harder instances, generated by the same instance generation algorithm. Furthermore, they generously shared the C++ code they used to achieve their results.

The test-instances are randomly generated instances, inspired by Dutch retail chains. The 60 instances contain 10 instances of 10, 15, 20, 25, 30 and 35 customers respectively, each with 3 demand scenarios. The demand scenarios correspond to high, medium and low demand, each with equal probability of occurrence. The average demand is about 1/6 truck load. Over 95% of the clients has demand between 1/15 and 1/3 truck load. The exogenous time windows are rather wide: on average the exogenous time window of the client has width 10.8, compared to an endogenous time window width of 2. See Spliet and Gabor (2014) for a detailed description.

All results are run on an Intel i7 3.5GHz computer with 16GB of RAM. Windows 7 is started in safe mode, running on a single thread on a single core. This removes the option to run code in parallel, yielding a fair comparison.

For the experiments, we only use the first 40 instances, which is the same set of instances used by Spliet and Gabor (2014). This allows us to use the 32 bit version of CPLEX, which is generally (slightly) faster than the 64 bit version but only allows up to 2GB of internal memory to be used. When we find the best possible settings for our branch-and-cut algorithm, we test its performance on the 20 new instances. Doing so does require the 64 bit version of CPLEX, which allows for the full 16GB of memory to be used.

6.2 Scope

To determine which experiments require an extensive analysis, we tried a variety of settings before conducting the actual analysis. It seems the following can be concluded:

Capacity cuts are required
Without adding the capacity cuts, even the smallest instances become virtually unsolvable, even when adding all other valid inequalities we discussed as well. This is consistent with the findings of the paper of Spliet and Gabor (2014), where the capacity cuts were also of major importance.

It is beneficial to add one-way arc constraints
As discussed earlier, the number of one-way arc constraints is sufficiently small such that all constraints can be added to the program at once. Adding these constraints strengthens the formulation and thereby reduces the number of nodes to be explored. Testing suggests this clearly compensates for the effect of the increase in the number of constraints. It is thus beneficial to add all one-way arc constraints at the beginning.

Preliminary testing was not decisive in which time constraints and capacity constraints to use, hence we will perform experiments to determine the best choices. Whatever choice we make regarding time and capacity constraints, when using branch-and-cut we will add the one-way arc constraints from the beginning and the capacity cuts at each node. The effect of the other two classes of inequalities, precedence inequalities and sub-optimal path inequalities, will also be subject to analysis.

In our branch-and-cut, we will always search for violated valid inequalities at each node of the branch-and-cut tree, until no more inequalities can be added. All classes of inequalities are added at once.

6.3 Experiments

We will do experiments to answers the following questions:

- Which combination of time and capacity constraints gives the best performance on the LP relaxation?
- Which combination of time and capacity constraints gives the best performance in branch-and-cut?
- How do these results compare to the results of Spliet and Gabor (2014)?
- Can precedence inequalities be used to speed up branch-and-cut?
- Can sub-optimal path inequalities be used to speed up branch-and-cut?
- Can we also solve larger instances?

The first question can be answered by considering the LP relaxation of the general flow formulation, for different choices of time and capacity constraints. If we compare LP objective values and runtime, combined with the theoretical comparisons in Sections 2.1.5 and 2.2.6, we can make statements regarding the performance of the different constraints on the LP relaxation.

Using these statements, we narrow down our options to be able to run branch-and-cut for the remaining options. We will do so without precedence inequalities and sub-optimal path inequalities, which will be tested later. This experiment will result in a definitive choice for both time and capacity constraints.

After deciding on the time and capacity constraints, we can compare our branch-and-cut results to the results of Spliet and Gabor (2014). Their code will be run on the same computer used to generate all other results, hence yielding a fair comparison.

Then, we will test whether our novel valid inequalities, the precedence inequalities and the sub-optimal path inequalities, speed up the branch-and-cut algorithm. We will do so by running our branch-and-cut algorithm for different settings.

The final question is whether our branch-and-cut algorithm is able to solve larger instances than what was previously possible. To this end, we run branch-and-cut on the 20 new instances and analyze the results.

7 Results

In this section, we will discuss the results obtained from the experiments laid out in the previous section. We will start with the analysis of the time and capacity constraints.

7.1 Analysis LP relaxation

To answer the question of which combination of time and capacity constraints gives the best performance on the LP relaxation, we solve the LP relaxation of the general flow formulation for different combinations of time and capacity constraint and present the objective values and the solution times.

We have implemented all the sets of time constraints that were discussed in Section 2.1. That is, MTZ inequalities (MTZ), affine combination predecessors (AC pred.), affine combination successors (AC succ.), affine combinations of both predecessors and succesors (AC both), the convex hull formulation (CH) and the improved convex hull formulation (ICH).

All time constraints are applicable to capacity as well. Furthermore, we have introduced capacity specific formulations in Section 2.2. To limit the number of possible combinations, we will only consider the two-commodity flow formulation (2CF), MTZ and ICH to model capacity. The advantages of 2CF have been explained in Section 2.2.6. MTZ was chosen to be applied to capacity because it requires only a small number of variables and constraints, potentially making it very fast. ICH was chosen because it is similar to the potentially strong CH, but requires less variables and constraints.

Table 8 shows the average objective value of the LP relaxation over the 40 test-instances for different combinations of constraints. 'None' indicates that no time constraints or capacity constraints were added to the general flow formulation. The non-aggregated data can be found in Tables 16-19 in the appendix.

Table 9 presents the average runtime in ms for the same combinations of constraints, over the same instances. The non-aggregated data on which Table 9 is based can be found in Tables 20-23 in the appendix.

If we consider the cases where the capacity constraints have been disabled (bottom row of Table 8 and Table 9), we can compare the strength of the different time constraints. The found objective values can be compared to each other, but not to the other objective values, as leaving out the capacity constraints yields a linear program that no longer corresponds to the TWAVRP.

The first five constraints seem pretty similar in terms of objective value, but the improved convex hull formulation clearly stands out with a better average objective value than the other time constraints. Furthermore, ICH gives the highest objective value for 39 of the 40 instances. This comes at a cost though: ICH is more than 20 times slower than MTZ.

The objective values can also be used to determine empirically between which

		Time constraints						
	Objective	MTZ	AC pred.	AC succ.	AC both	CH	ICH	None
Capacity	MTZ	15.17	15.18	15.17	15.19	15.18	16.26	15.17
	2CF	20.92	20.92	20.92	20.92	20.92	20.92	20.92
	ICH	21.12	21.12	21.12	21.12	21.12	21.12	21.12
	None	14.61	14.64	14.63	14.71	14.65	15.81	14.22

Table 8: Average objective value for the LP relaxation of the general flow formulation, for various time and capacity constraints, instances 1-40

		Time constraints						
	Runtime	MTZ	AC pred.	AC succ.	AC both	CH	ICH	None
Capacity	MTZ	6	11	45	51	39	139	3
	2CF	25	56	83	126	120	101	21
	ICH	45	62	94	132	133	105	35
	None	3	11	36	54	43	78	2

Table 9: Average runtime in ms for the LP relaxation of the general flow formulation, for various time and capacity constraints, instances 1-40

time constraints there does *not* exist a dominance relationship. E.g. if MTZ has a lower objective value than ICH for one instance, but a higher objective value for another instance, than MTZ and ICH clearly do not have a dominance relationship. The results are presented in Table 10.

Most of the relationships for which we could not make empirical statements, have already been proven to be dominant: convex hull is stronger than MTZ, and combining affine combinations predecessors with successors results in a formulation that is stronger than either predecessors or successors alone. Only one relation is yet unaccounted for: the LP results suggest that ICH always gives better objective values than AC successors. It could be interesting to prove or disprove this statement, but in this thesis, we will not consider this further.

The question is whether the differences in stength for the different time constraints persist when we consider the capacity constraints as well. The important role of the capacity cuts already suggests that the capacity constraints are more important for the strength of the general flow formulation than the time constraints. This can also be seen from Table 8: when 2CF or ICH are chosen to model capacity, the choice of time constraints is almost irrelevant for the LP objective value. Even removing the time constraints completely (last column) only yields a minimal decrease in objective value.

If we choose to model capacity by 2CF or ICH, we are thus best off by choosing the time constraints that allows for the quickest LP solution. Table 9 shows that this is clearly MTZ (time) in the average case, and Tables 20-23 support this conclusion per instance.

If we choose to model capacity by MTZ (capacity), the objective values for

	MTZ	AC pred.	AC succ.	AC both	CH	ICH
MTZ		X	X	X		X
AC pred.			X		X	X
AC succ.					X	
AC both					X	X
CH						X
ICH						

Table 10: Combinations of time constraints for which it could be shown empirically that there does *not* exist a dominance relationship, indicated by X

different time constraints are also similar, execpt for ICH (time) which gives a higher objective value. The combination of MTZ (capacity) and ICH (time) is not likely to be effective though, as it is slower than the combination of 2CF and MTZ (time), and also has a lower objective value. The only combination with MTZ (capacity) worth considering is the combination with MTZ (time): the objective value may be relatively bad, but the speed of this combination might make up for that in branch-and-cut.

We can conclude that in a branch-and-cut framework, modeling time with the MTZ inequalities is likely to be the most successful. We have access to stronger time constraints, but the difference in strength disappears when capacity constraints are added. For this reason, it is only logical to use the fastest constraints.

Regarding the capacity constraints we can not yet make a decision, as MTZ, 2CF and ICH have both differences in LP objective values and in solution time. From Tables 8 and 9 alone, it is impossible to say which capacity constraints will be best in branch-and-cut. We will thus have to use all three sets of capacity constraints in branch-and-cut to determine which one is best.

7.2 Capacity constraints in branch-and-cut

In the previous sections we have analyzed the time and capacity constraints with experiments on the LP relaxation of the general flow formulation. We concluded that we would use the MTZ inequalities to model time. To model capacity, however, we could not make a decision between the MTZ inequalities, the 2-commodity flow formulation and the improved convex hull formulation.

In this section, we will compare these three different ways to model capacity in a branch-and-cut algorithm. Time will be modeled by MTZ. As explained in Section 6.2 we will add all one-way arcs constraints from the beginning and add capacity cuts during the optimization. For this experiment, no precedence cuts or sub-optimal path inequalities will be added.

The results of this experiment are presented in Table 11. 'Runtime' refers to the total time necessary to solve the instance of the TWAVRP to optimality. 'Processed nodes' is the number of nodes in the branch-and-cut tree that were explored. 'Root gap' indicates the percentual difference between the lower bound in the root node (after adding cuts) and the best found upper bound. If CPLEX processes the root node multiple times, the lower bound is used from the first time the root node was processed.

It can be seen that MTZ is pretty competitive for certain instances, though 2CF and ICH seems to perform better overall. MTZ is characterized by a large number of processed nodes: over 60% more than necessary for 2CF. Frankly, the root gaps of MTZ, 2CF and ICH are similar. This implies that in the root node, the capacity cuts make up for the weakness of MTZ, though this does not happen in the rest of the tree, as still a lot of branching is required.

Choosing between 2CF and ICH is difficult: both use a similar amount of time (807 seconds versus 931 seconds) and nodes (160,000 versus 180,000) to solve all instances to optimality. Figure 11 shows a logaritmic plot of the runtime of MTZ, 2CF and ICH over the instances. From this plot it becomes clear that though ICH is often faster than 2CF, 2CF is usually faster when instances take long to solve, most notably instances 12, 33 and 36. If we consider solving even harder instances, this is certainly an advantage.

We thus choose to use the 2-commodity flow formulation in our branch-and-cut algorithm. 2CF gives good bounds and yields lower computation times for larger instances.

7.3 Comparison with previous results

Now that we have decided both on which time constraints and on which capacity constraints to use, we will compare the performance of our branch-and-cut algorithm with capacity cuts and one-way arc inequalities to the performance of the branch-price-and-cut algorithm of Spliet and Gabor (2014).

To this end, we run the same computer program on the same set of instances, both provided by Spliet and Gabor, on the same computer that is used to create all other results in this thesis. The results are thus directly comparable. The results we present are based on the so called branch-price-and-cut with 2-cycle elimination, which is the solution method of Spliet and Gabor (2014) yielding best average time performance on the test set.

The results per instance can be found in Table 12 and a plot comparing the solution times can be found in Figure 12.

What immediately stands out is the enormous decrease in total runtime. With their approach, Spliet and Gabor require over 10 hours of runtime to pro-

	Runtime			**Processed nodes**			**Root gap**		
Inst.	**MTZ**	**2CF**	**ICH**	**MTZ**	**2CF**	**ICH**	**MTZ**	**2CF**	**ICH**
1	0.0	0.1	0.0	1	1	1	0	0	0
2	0.3	0.4	0.1	87	193	29	0.28	0.28	0.28
3	0.0	0.1	0.1	1	1	1	0	0	0
4	0.1	0.3	0.3	78	141	95	0.14	0.14	0.14
5	0.1	0.5	0.1	35	163	1	0.34	0.34	0.34
6	0.0	0.1	0.0	1	1	1	0	0	0
7	0.0	0.0	0.0	1	1	1	0	0	0
8	0.2	0.4	0.2	149	201	14	0.96	0.96	0.96
9	0.0	0.0	0.1	1	1	1	0	0	0
10	0.0	0.3	0.1	1	105	1	0	0	0
11	0.1	0.1	0.1	1	1	1	0	0	0
12	211.6	114.7	155.3	128278	67606	89143	2.36	2.36	2.36
13	8.5	20.7	12.7	5847	12450	6662	1.78	1.78	1.78
14	1.1	1.9	1.1	291	332	196	0.03	0.03	0.03
15	0.2	2.0	1.1	18	302	151	0.17	0.17	0.17
16	1.3	0.7	0.4	350	144	5	0.17	0.17	0.17
17	0.1	0.2	0.2	1	1	1	0	0	0
18	0.3	5.1	1.7	66	2078	542	0.47	0.47	0.47
19	1.6	2.7	0.7	728	756	66	0.97	0.97	0.97
20	1.0	0.2	0.3	178	1	1	0	0	0
21	14.3	1.4	1.5	2655	121	77	1.11	1.11	1.11
22	3.6	8.3	8.5	601	813	978	0.19	0.19	0.19
23	3.6	0.5	0.5	605	1	1	0.15	0.12	0.12
24	3.4	4.5	5.1	605	810	662	0.86	0.86	0.86
25	10.5	17.9	20.4	3336	4947	4520	1.07	1.08	1.08
26	0.4	0.4	0.5	17	1	1	0	0	0
27	3.7	1.3	0.7	411	19	1	0	0	0
28	0.7	1.0	1.6	33	34	24	0.08	0.08	0.08
29	0.6	0.7	0.7	22	7	7	0.05	0.05	0.05
30	0.2	0.3	0.4	1	1	1	0	0	0
31	24.1	27.5	13.5	2796	2398	1172	0.57	0.57	0.57
32	1.3	6.3	3.9	34	591	161	0.27	0.27	0.27
33	100.8	97.1	135.1	16539	9704	16067	1.03	1.03	1.03
34	13.2	6.8	29.4	1553	467	2567	0.33	0.33	0.33
35	35.4	62.7	25.8	5693	6460	2753	0.83	0.82	0.83
36	624.5	265.3	334.3	70041	31407	32984	1.49	1.51	1.58
37	58.9	15.8	23.7	10378	5906	6495	0.43	0.43	0.46
38	58.3	73.4	107.1	7079	9085	7671	0.59	0.59	0.59
39	20.0	24.0	36.2	2163	2191	3641	0.94	0.94	0.94
40	6.7	41.1	7.3	1595	2516	610	0.62	0.63	0.62
Total	1211.0	806.9	930.8	262270	161958	177306	18.3	18.3	18.4

Table 11: Branch-and-cut results for various capacity constraints

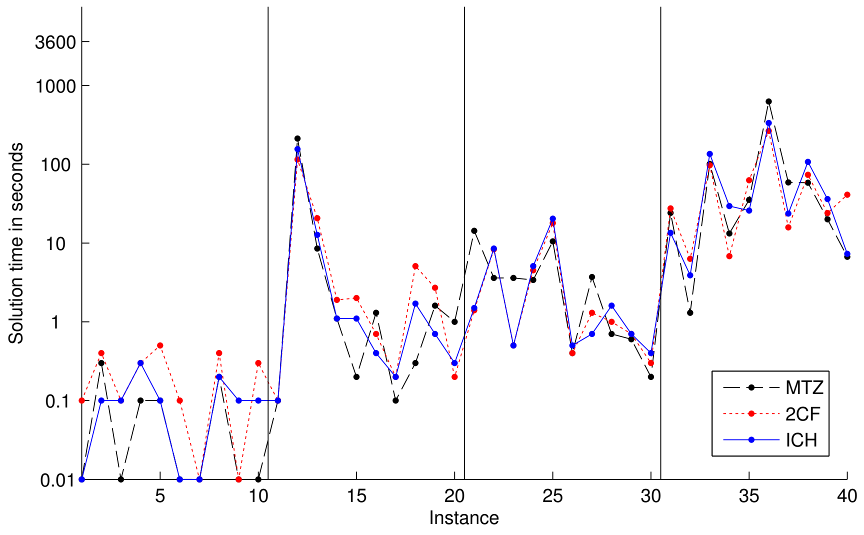

Figure 11: Logaritmic plot comparing the total solution time of branch-and-cut for three sets of capacity constraints

cess all instances, while the current branch-and-cut algorithm manages to do so in under 15 minutes. That is a decrease in runtime of almost 98%. Furthermore, all eight instances that previously could not be solved within one hour of computation time, have now been solved. The difference in runtime is also clearly visible in Figure 12: for all instances, branch-and-cut is magnitudes faster than branch-price-and cut.

Table 12 also provides an explanation for the speedup: the number of processed nodes of branch-and-cut is almost 15 times that of branch-price-and-cut. Furthermore, on average the root gap for branch-and-cut is almost twice as big as the root gap for branch-price-and-cut. In other words: branch-price-and-cut gives very strong bounds, but requires a relatively slow pricing algorithm to be executed at each node. It turns out that using branch-and-cut with a flow formulation makes up for the weaker bounds by processing nodes faster.

7.4 Analysis precedence inequalities

In this section, we will test how well the precedence inequalities perform in the branch-and-cut algorithm. We have discussed three ways to separate precedence inequalities: separation by paths, separation by DAGs and exact separation.

When we find precedence inequalities using separation by paths, we also test two additional strategies. The first one is to convert both candidates making up the inequality to DAGs, as explained in Section 4.2.4 and visualized by Figures 8 and 9. The second strategy is to convert candidates to candidates containing a complete subgraph, as explained in Section 4.2.4. To do so, we use Theorem 9

	Runtime		**Processed nodes**		**Optimality gap**		**Root gap**	
Inst.	**BP&C**	**B&C**	**BP&C**	**B&C**	**BP&C**	**B&C**	**BP&C**	**B&C**
1	0.7	0.1	1	1	0	0	0	0
2	121.9	0.4	483	193	0	0	0.17	0.28
3	3.7	0.1	1	1	0	0	0	0
4	28.5	0.3	193	141	0	0	0.14	0.14
5	2.3	0.5	2	163	0	0	0	0.34
6	1.5	0.1	2	1	0	0	0	0
7	4.9	0.0	4	1	0	0	0	0
8	3.5	0.4	29	201	0	0	0.65	0.96
9	3.0	0.0	7	1	0	0	0	0
10	5.9	0.3	5	105	0	0	0	0
11	87.4	0.1	22	1	0	0	0	0
12	3600.0	114.7	889	67606	0.15*	0	0.67*	2.36
13	3600.0	20.7	684	12450	0.59*	0	1.10*	1.78
14	58.0	1.9	45	332	0	0	0	0.03
15	29.4	2.0	36	302	0	0	0	0.17
16	92.4	0.7	98	144	0	0	0.10	0.17
17	22.9	0.2	15	1	0	0	0	0
18	105.3	5.1	98	2078	0	0	0.20	0.47
19	133.3	2.7	133	756	0	0	0.56	0.97
20	41.6	0.2	28	1	0	0	0	0
21	3600.0	1.4	864	121	0.02*	0	0.57*	1.11
22	152.3	8.3	62	813	0	0	0.03	0.19
23	99.6	0.5	40	1	0	0	0	0.12
24	112.2	4.5	27	810	0	0	0.03	0.86
25	3600.0	17.9	712	4947	0.08*	0	0.61*	1.08
26	65.4	0.4	16	1	0	0	0	0
27	85.5	1.3	24	19	0	0	0	0
28	106.5	1.0	36	34	0	0	0	0.08
29	65.5	0.7	17	7	0	0	0	0.05
30	45.1	0.3	4	1	0	0	0	0
31	610.9	27.5	121	2398	0	0	0.13	0.57
32	840.0	6.3	164	591	0	0	0.07	0.27
33	3600.0	97.1	413	9704	0.33*	0	0.45*	1.03
34	193.2	6.8	36	467	0	0	0	0.33
35	640.0	62.7	119	6460	0	0	0	0.82
36	3600.0	265.3	1662	31407	0.13*	0	0.43*	1.51
37	3600.0	15.8	278	5906	0.29*	0	0.29*	0.43
38	3600.0	73.4	1259	9085	0.12*	0	0.30*	0.59
39	3600.0	24.0	2294	2191	0	0	0.50	0.94
40	1093.2	41.1	521	2516	0	0	0.30	0.63
Total	37255.3	806.9	11444	161958	1.72	0	7.33	18.28

Table 12: Branch-price-and-cut with 2-cycle elimination (BP&C) of Spliet and Gabor (2014) compared to our branch-and-cut (B&C) using MTZ to model time and 2CF to model capacity. Runtimes are in seconds. An * indicates the value is calculated using the optimal solution of B&C

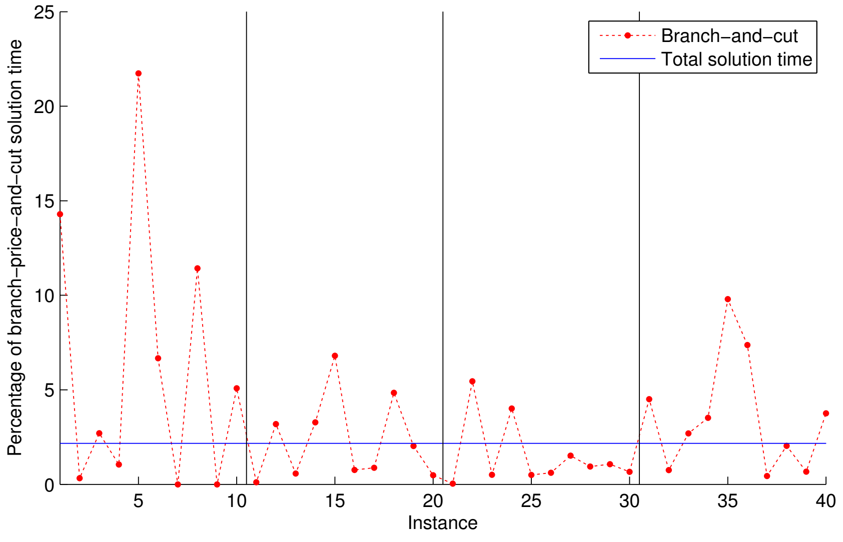

Figure 12: Plot showing the differences in solution time between branch-and-cut and branch-price-and-cut as implemented by Spliet and Gabor (2014). 'Total solution time' shows the result for the aggregated solution times.

to calculate a lower bound on the time necessary to visit all nodes. If the new lower bound does not allow for a stronger precedence inequality to be found, conversion to DAGs is used.

After finding precedence inequalities using separation by DAGs, we also apply this second strategy. That is, the candidates are converted to contain complete subgraphs, if possible.

Note that there exist more sophisticated acceleration strategies. One could, for example, adjust the algorithm for separation by paths to perform the conversion to DAGs as a part of the separation algorithm. When done correctly, this can increase the number of found inequalities. We will, however, not consider additional acceleration strategies in this thesis. The advantage of the acceleration strategies we use for the experiments, is that they are independent of the separation algorithm. That is, when a better separation algorithm is found, they can still be used.

The result of the experiments can be found in Table 13. 'Path', 'DAG' and 'Compl.' indicate the inequalities are not converted, converted to DAGs, or converted to contain complete subgraphs, respectively. Note that exact separation is not included in the table, as it turned out to be too slow to use in branch-and-cut.

The table shows that using precedence inequalities is a competitive alternative to branch-and-cut without precedence inequalities. It is difficult to draw conclusions based on the total runtimes, as they are largely influenced by the

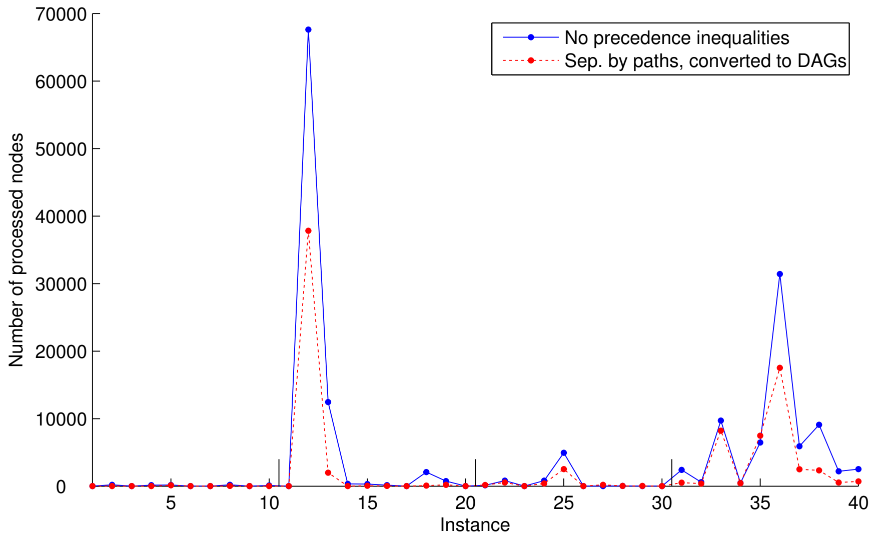

Figure 13: Plot comparing the number of processed nodes without precedence inequalities, and when using separation by paths and conversion to DAGs

difficult instances. For example, the differences in total runtime are mainly caused by instance 36, which is relatively hard to solve.

A good argument for using the precedence inequalities is the reduction in the number of processed nodes. For each tested setting, the total number of processed nodes was less than the total number of processed nodes when no precedence inequalities were added. Furthermore, separation by paths and conversion to DAGs yielded a total number of processed nodes almost half of the total number of processed nodes necessary when no precedence inequalities are being used. This is shown graphically in Figure 13.

If we consider the different acceleration strategies, we would expect that they would cause a decrease in the number of processed nodes. After all, precedence inequalities get replaced by stronger precedence inequalities. We indeed see this effect for separation by paths with conversion to DAGs and for separation by DAGs with conversion to complete graphs. In both cases, the total number of processed nodes and the total runtime is lower than in the case where the precedence inequalities are not converted.

Separation by paths with conversion to complete graphs, however, is an exception: instead of the expected decrease we see an enormous increase in the total runtime and the total number of processed nodes. A possible explanation could be bad luck: each constraint can change the order in which the branch-and-cut tree is traversed, which can have a big effect on both runtime and the number of processed nodes, causing an outlier. Instance 36 could be such an outlier, as the number of nodes (79,739) is very big when compared to that of the run without conversion of inequalities (37,639 nodes).

Using the exact separation algorithm we could not run the branch-and-cut algorithm to completion, as it was too slow. Instead we performed branch-and-cut on the root node only. That is, we iterate between solving the LP relaxation and adding capacity and precedence cuts, but we do not branch. We did so for the same settings as in Table 13, plus exact separation.

The experiment resulted in lower bounds for the objective value that were almost equal for all settings. The maximum difference per instance over different settings was only 0.03% of the objective value. Further testing revealed that the lower bounds do not even change when the precedence inequalities are disabled. This is surprising: from Table 13 we know that the precedence inequalities have effect, but in the root node we do not see an increase in lower bound when they are added. Apparently, the precedence inequalities become more important deeper in the tree.

This could be explained by the nature of the precedence inequalities: they disallow certain combinations of paths. If a solution is very fractional, not many paths can be detected, and hence the precedence inequalities are of little use. Deeper in the tree, where many variables are fixed, they become more effective.

We conclude by stating there is a big potential in the precedence inequalities. They are able to reduce the number processed nodes considerably, while still being competitive. This implies further speedups may be achieved by considering different (heuristic) separation algorithms and acceleration strategies. Also adding only a subset of the precedence inequalities, e.g. only cuts with at least a violation of 0.5, could yield results. Though we will not consider this further in this thesis, it is certainly an interesting path for further research.

7.5 Analysis sub-optimal path inequalities

In the previous section, we have seen that the fastest way to solve all 40 test-instances, is by adding precedence inequalities separated by paths and converted to DAGs.

To test the effect of the sub-optimal path inequalities, we run two experiments: one with the precedence inequalities as mentioned above, and one without precedence inequalities. The results are presented in Table 14.

The sub-optimal path inequalities do not seem very effective: we do not see a decrease in the total runtime, nor a decrease in the total number of processed nodes when compared to the previous experiments. Still, adding sub-optimal path inequalities is competitive on certain instances, most notably instance 36, which allows for a reasonably fast solution when no precedence cuts are added.

A possible explanation for the bad performance of the sub-optimal path inequalities has to do with the settings of CPLEX. The sub-optimal path inequalities allow for feasible solutions to the TWAVRP to be cut off. CPLEX classifies such optimality cuts as ‘lazy constraints’, as opposed to valid inequalities that do not cut off feasible solutions, which are called ‘user cuts’. Lazy constraints require some CPLEX presolve reductions to be turned off, which may be hindering the performance of the sub-optimal path inequalities.

7.6 Larger instances

In the previous experiments, we have found that branch-and-cut with precedence inequalities separated by paths and converted to DAGs but without sub-optimal-path inequalities is our fastest method to solve the first 40 test-instances. We are now able to solve all 40 test-instances to optimality, in under 10 minutes total.

The first 40 test-instances contained at most 25 customers and 3 scenarios. To find out the maximum size of the instances we can solve, we test our fastest branch-and-cut algorithm on 20 additional instances with 30 and 35 customers, generated by the same algorithm that was used to create the original 40 test-instances. The results are presented in Table 15.

It is no surprise that the larger instances are much harder to solve; run-time is expected to increase exponentially in the number of customers. What is surprising is that we can solve almost half of the new instances to optimality: 9 of the 20 instances were solved to optimality within one hour of computation time. The solutions for the other instances are all close to optimality, the largest optimality gap being just over 3%. With such a small optimality gap, even the sub-optimal solution of the TWAVRP potentially outperforms current practice.

For even larger instances, heuristics are still required. Note that in Table 15 the root gaps are all below 4%, indicating that processing only the root node could be a good starting point for a heuristic.

	Runtime					**Processed nodes**				
	Sep. by paths			**Sep. by DAGs**		**Sep. by paths**			**Sep. by DAGs**	
Inst.	**Path**	**DAG**	**Compl.**	**DAG**	**Path**	**Path**	**DAG**	**Compl.**	**DAG**	**Compl.**
1	0.2	0.0	0.0	0.0	0.0	1	1	1	1	1
2	0.2	0.2	0.2	0.2	0.1	35	21	18	11	13
3	0.1	0.1	0.1	0.1	0.1	1	1	1	1	1
4	0.2	0.1	0.1	0.1	0.1	15	5	5	5	5
5	0.2	0.5	0.1	0.5	0.3	22	124	3	86	39
6	0.0	0.1	0.0	0.0	0.0	1	1	1	1	1
7	0.0	0.0	0.0	0.0	0.0	1	1	1	1	1
8	0.3	0.3	0.3	0.3	0.3	38	38	38	38	38
9	0.0	0.0	0.0	0.0	0.0	1	1	1	1	1
10	0.1	0.0	0.1	0.1	0.0	1	1	1	1	1
11	0.1	0.1	0.1	0.1	0.1	1	1	1	1	1
12	169.2	126.3	170.9	131.3	190.3	57314	37813	54681	39830	53590
13	5.8	5.1	9.3	5.8	5.8	2407	2001	3694	2126	2248
14	0.4	0.3	0.3	0.3	0.5	9	7	7	12	38
15	0.6	0.6	0.3	0.5	0.3	57	60	15	42	15
16	0.8	0.5	1.3	0.5	1.3	73	37	158	37	155
17	0.2	0.2	0.2	0.2	0.2	1	1	1	1	1
18	3.0	1.1	1.5	1.4	1.4	450	92	156	240	181
19	1.1	1.1	1.3	1.1	1.2	196	196	235	196	216
20	0.3	0.3	0.2	0.3	0.2	14	17	1	17	1
21	2.1	1.3	8.0	1.7	2.2	305	169	579	224	237
22	2.6	8.1	4.0	8.3	4.8	91	552	154	528	246
23	0.6	0.6	0.5	0.6	0.5	1	1	1	1	1
24	2.4	2.7	1.6	3.7	1.8	339	409	178	477	186
25	12.4	13.5	13.0	18.1	21.2	1687	2517	2006	2921	3749
26	0.3	0.3	0.3	0.3	0.3	1	1	1	1	1
27	1.4	2.6	1.4	2.3	1.3	11	207	20	137	18
28	2.2	1.4	6.6	2.1	1.9	122	30	540	124	122
29	1.0	1.1	1.1	1.3	1.5	14	36	21	51	91
30	0.3	0.4	0.3	0.4	0.3	1	1	1	1	1
31	13.2	10.9	16.0	11.3	13.7	762	512	1086	741	1097
32	2.0	6.8	3.3	3.2	2.2	81	390	111	64	28
33	64.1	85.5	83.7	101.9	59.0	7286	8210	7753	8824	5170
34	7.7	7.0	8.0	24.2	18.2	520	421	423	1471	1248
35	51.8	75.9	38.6	57.2	34.0	4841	7474	4069	5865	3262
36	331.6	173.4	1036.3	946.6	525.9	37639	17523	79739	74661	35975
37	7.1	16.4	5.6	7.8	4.8	816	2504	774	1275	627
38	45.7	36.9	19.2	31.6	12.9	2274	2341	1188	1252	755
39	10.6	8.0	11.9	63.5	26.3	862	536	885	5164	1548
40	13.7	10.0	7.1	27.1	8.1	860	711	748	1734	794
Total	755.7	599.7	1453.0	1455.9	943.2	119151	84964	159296	148164	111703

Table 13: Runtime (in seconds) and number of processed nodes for branch-and-cut with precedence inequalities, for two separation algorithms and various acceleration strategies.

	Runtime		**Processed nodes**		**Optimality gap**	
Inst.	**No prec.**	**Prec.**	**No prec.**	**Prec.**	**No prec.**	**Prec.**
1	0.0	0.0	1	1	0	0
2	0.5	0.2	355	41	0.28	0.28
3	0.0	0.0	1	1	0	0
4	1.7	0.3	1254	19	0.14	0.14
5	0.6	0.3	176	46	0.34	0.34
6	0.1	0.0	1	1	0	0
7	0.0	0.0	1	1	0	0
8	0.1	0.2	21	42	0.96	0.96
9	0.1	0.0	1	1	0	0
10	0.4	0.1	118	1	0	0
11	3.0	0.1	605	1	0	0
12	667.1	116.0	263771	36826	2.36	2.36
13	31.3	10.4	15531	3628	1.78	1.78
14	3.0	0.4	603	10	0.03	0.03
15	3.0	0.3	587	15	0.17	0.17
16	0.3	0.5	19	46	0.17	0.17
17	0.2	0.2	1	1	0	0
18	3.5	3.4	1013	408	0.47	0.47
19	2.7	2.3	820	556	0.97	0.97
20	1.7	0.2	262	1	0	0
21	1.4	1.2	157	114	1.11	1.11
22	14.2	3.6	1004	151	0.19	0.19
23	8.2	0.9	1001	15	0.15	0.12
24	10.2	3.1	1916	242	0.86	0.86
25	16.2	9.9	3055	1543	1.07	1.07
26	3.0	0.2	201	1	0	0
27	0.5	6.3	11	411	0	0
28	1.0	1.0	28	27	0.08	0.08
29	8.2	5.1	878	296	0.05	0.05
30	0.3	0.9	1	25	0	0
31	96.9	36.1	9681	2965	0.57	0.57
32	3.5	2.8	137	141	0.27	0.27
33	155.4	55.3	17496	6182	1.03	1.03
34	36.3	35.4	2230	1643	0.33	0.33
35	93.3	166.6	7180	11888	0.82	0.82
36	196.0	576.3	19516	54735	1.49	1.49
37	253.2	11.2	26772	962	0.43	0.43
38	69.9	88.5	6727	4599	0.59	0.59
39	41.4	54.8	3739	3889	0.94	0.94
40	18.8	9.5	1497	861	0.63	0.62
Total	1747.3	1203.7	388368	132336	18.28	18.24

Table 14: Result for branch-and-cut with sub-optimal path inequalities. 'No prec.' indicates no precedence inequalities are added. 'Prec.' indicates precedence inequalities were separated by paths and converted to DAGs.

Inst.	**Runtime**	**Processed nodes**	**Optimality gap**	**Root gap**
41	3600.0	124301	0.81	2.00
42	3600.0	92361	2.62	3.56
43	3600.0	77101	1.20	2.01
44	263.4	11253	0	1.46
45	3600.0	151501	0.59	1.82
46	31.7	2186	0	0.57
47	188.7	10063	0	0.68
48	3600.0	85201	0.42	1.86
49	3600.0	90701	1.08	2.42
50	300.7	14160	0	0.71
51	168.8	4695	0	0.76
52	10.7	79	0	0.06
53	3600.0	45393	2.98	3.30
54	3600.0	50100	2.69	3.42
55	231.3	3557	0	0.67
56	3600.0	51370	3.01	3.69
57	3600.0	41976	2.63	3.10
58	152.5	8729	0	0.88
59	1480.8	54357	0	0.93
60	3601.0	83501	0.18	1.32
Total	42430.3	1002585	18.22	35.23

Table 15: Results for branch-and-cut with precedence inequalities separated by paths and converted to DAGs, without sub-optimal path inequalities.

8 Conclusion

In this thesis, we have taken a branch-and-cut approach to solving the TWAVRP, using flow formulations. Testing different combinations of both known and novel time constraints and capacity constraints, we have determined that we get the best results when modeling time with the Miller-Tucker-Zemlin inequalities, and capacity with the 2-commodity flow formulation of Baldacci et al. (2004).

We have used the package of Lysgaard (2003) to add capacity cuts, which was of major importance for the speed of the branch-and-cut algorithm. Furthermore, we have added one-way arc constraints. After adding both valid inequalities, the branch-and-cut algorithm outperforms the branch-price-and-cut algorithm of Spliet and Gabor (2014) by solving all test-instances to optimaly in only 2.2% of the time required by the branch-price-and-cut algorithm.

Testing two novel valid inequalities: the precedence inequalities and the suboptimal path inequalities, we found the former to be effective and the latter to be uneffective. The precedence inequalities allow for a vast reduction in the number of nodes that have to be processed. Using the 'separation by paths' separation algorithm, and applying the 'convert to DAGs' acceleration strategy, we were able to boost performance even further. Solving all instances of the test set now only requires just under 10 minutes.

Having solved all test-instances, we generated 20 harder test-instances and tried to solve them as well. We were able to solve 9 instances to optimality within one hour of computation time per instance. For the other 11 instances, the branch-and-cut algorithm generated feasible solutions with objectives at most 3% from the optimal objective.

8.1 Explanation of performance difference

The big difference in performance between the branch-and-cut algorithm of this thesis and the branch-price-and-cut algorithm of Spliet and Gabor (2014) is rather surprising. Recently, column generation has been applied to many VRP like problems, and it is to be expected that branch-price-and-cut and branch-and-cut are competitive.

One explanation for the differences in solution time is in the implementation of both algorithms. For the branch-and-cut algorithm, we have used commercial solver CPLEX as the basis, while Spliet and Gabor (2014) have programmed the branch-price-and-cut algorithm themselves, and rely on CPLEX solely to solve the LP relaxations. Using CPLEX to manage the branch-and-cut brings many advantages, including advanced branching strategies, upper bound heuristics, efficient tree management, efficient cut pools and presolver reductions. Their combined effect could explain the differences in speed between the two algorithms.

A major difference between branch-price-and-cut and branch-and-cut, is that branch-price-and-cut uses a relatively large amount of time per node to calculate a strong lower bound, while the branch-and-cut algorithm calculates a weaker lower bound very fast. This could explain the differences as well: if branch-

and-cut can explore more nodes in the same time, it can still be faster than branch-price-and-cut.

8.2 Further research

The TWAVRP is a problem that has only been introduced recently and has plenty of opportunities for research, both motivated by practice and by theory.

In this thesis, we have only considered solving the TWAVRP to optimality. We succeed in doing so for up to about 35 customers and 3 scenarios. Though impressive considering the complexity of the TWAVRP, planning up to 35 customers may not be sufficient in practice. Hence, it may be necessary to look into heuristics that can solve large instances of the TWAVRP to near-optimality in reasonable time. This could be either heuristics based on the VRP literature, meta-heuristics, or heuristics designed specifically for the TWAVRP.

Another way to make the TWAVRP better suitable for practical use, is to relax some of the assumptions of the TWAVRP. Examples could be including truck-unloading times, planning non-homogeneous fleets or considering time-dependent travel times to model busy traffic.

From a theoretical viewpoint, it is interesting to further develop the exact methods to solve the TWAVRP. Just as in the TSP and VRP literature, this is likely to increase the size of the problems that are solvable in reasonable time.

At the moment of writing, the branch-and-cut algorithm as described in this thesis is the fastest algorithm to solve the TWAVRP test-instances. We have explained that possibly an important part of the speed-up is caused by features of CPLEX. It would thus be interesting to see which features exactly cause this speed-up, and if we can use this information to improve the exact solution methods for the TWAVRP.

Another interesting research topic concerns the precedence inequalities. We have shown that adding precedence inequalities yield a big reduction in the number of nodes that have to be processed. If we can improve upon the separation algorithms, we might improve on performance. Another interesting tactic would be to only add precedence cuts that are violated by a minimum amount.

The precedence inequalities can be applied to branch-price-and-cut as well, potentially improving performance. A direct application of Theorem 1 could be even more successful: Theorem 1 explains which paths directly exclude each other. In this thesis, we have used Theorem 1 to derive the precedence inequalities, but when using the branch-price-and-cut algorithm we can directly add constraints on paths to exclude each other. Such a branch-price-and-cut specific valid inequality might prove very effictive.

References

Norbert Ascheuer, Matteo Fischetti, and Martin Grötschel. Solving the asymmetric travelling salesman problem with time windows by branch-and-cut. *Mathematical Programming*, 90(3):475–506, 2001.

Roberto Baldacci, Eleni Hadjiconstantinou, and Aristide Mingozzi. An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Operations Research*, 52(5):723–738, 2004.

Roberto Baldacci, Nicos Christofides, and Aristide Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115(2):351–385, 2008.

Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research*, 218(1):1–6, 2012.

Michel L Balinski and Richard E Quandt. On an integer program for a delivery problem. *Operations Research*, 12(2):300–304, 1964.

Jonathan F Bard, George Kontoravdis, and Gang Yu. A branch-and-cut procedure for the vehicle routing problem with time windows. *Transportation Science*, 36(2):250–269, 2002.

Ricardo Fukasawa, Humberto Longo, Jens Lysgaard, Marcus Poggi de Aragão, Marcelo Reis, Eduardo Uchoa, and Renato F Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical programming*, 106(3):491–511, 2006.

Ignacio E Grossmann and Juan P Ruiz. Generalized disjunctive programming: A framework for formulation and alternative algorithms for minlp optimization. In *Mixed Integer Nonlinear Programming*, pages 93–115. Springer, 2012.

Brian Kallehauge. Formulations and exact algorithms for the vehicle routing problem with time windows. *Computers & Operations Research*, 35(7):2307–2330, 2008.

Gilbert Laporte. Fifty years of vehicle routing. *Transportation Science*, 43(4): 408–416, 2009.

Gilbert Laporte, Yves Nobert, and Martin Desrochers. Optimal routing under capacity and distance restrictions. *Operations research*, 33(5):1050–1073, 1985.

Sangbum Lee and Ignacio E Grossmann. New algorithms for nonlinear generalized disjunctive programming. *Computers & Chemical Engineering*, 24(9): 2125–2141, 2000.

Jens Lysgaard. CVRPSEP: A package of separation routines for the Capacitated Vehicle Routing Problem. Working paper, Aarhus School of Business, 2003.

Jens Lysgaard, Adam N Letchford, and Richard W Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100(2):423–445, 2004.

Clair E Miller, Albert W Tucker, and Richard A Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7 (4):326–329, 1960.

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

Ramesh Raman and Ignacio E Grossmann. Modelling and computational techniques for logic based integer programming. *Computers & Chemical Engineering*, 18(7):563–578, 1994.

Paul Rubin. Binary Variables and Quadratic Terms, 2010. URL `http://orinanobworld.blogspot.nl/2010/10/binary-variables-and-quadratic-terms.html`.

Nicolas Sawaya and Ignacio Grossmann. Reformulations, relaxations and cutting planes for linear generalized disjunctive programming, 2008.

Remy Spliet and Guy Desaulniers. The discrete time window assignment vehicle routing problem. Working paper, Les Cahiers du GERAD G-2012-81, 2012.

Remy Spliet and Adriana F Gabor. The time window assignment vehicle routing problem. *Transportation Science*, page forthcoming, 2014.

Paolo Toth and Daniele Vigo. *The vehicle routing problem.* Siam, 2001.

A Simplified convex hull formulation

Note that in (31)-(34) multiple sets of v-variables only occur as a sum. These variables can thus be aggregated to reduce the number of variables and constraints. Aggregating $\sum_{i \in V' \cup \{0\}, i \neq l} v_l^{\omega ij}$ to ϕ_{lj}^{ω} $\forall j \in V', j \neq l, l \in V' \cup \{0\}, \omega \in \Omega$ yields:

Time constraints: convex hull formulation

$$t_l^{\omega} = \begin{cases} \sum_{i \in V' \cup \{0\}} v_l^{\omega il} & \text{if } l = j \\ v_l^{\omega lj} + \phi_{lj}^{\omega} & \text{if } l \neq j \end{cases} \quad \forall j \in V', l \in V' \cup \{0\}, \omega \in \Omega \tag{87}$$

$$v_j^{\omega ij} - v_i^{\omega ij} \geq t_{ij} x_{ij}^{\omega} \ \forall i \in V' \cup \{0\}, j \in V', \omega \in \Omega \tag{88}$$

$$s_l x_{ij}^{\omega} \leq v_l^{\omega ij} \leq e_l x_{ij}^{\omega} \ \forall i \in V' \cup \{0\}, j \in V', l \in \{i, j\}, \omega \in \Omega \tag{89}$$

$$\sum_{i \in V' \cup \{0\}, i \neq l} s_l x_{ij}^{\omega} \leq \phi_{lj}^{\omega} \leq \sum_{i \in V' \cup \{0\}, i \neq l} e_l x_{ij}^{\omega} \ \forall j \in V', l \in V' \cup \{0\}, l \neq j, \omega \in \Omega \tag{90}$$

$$v_l^{\omega ij} \geq 0 \ \forall i \in V' \cup \{0\}, j \in V', l \in \{i, j\}, \omega \in \Omega \tag{91}$$

$$\phi_{lj}^{\omega} \geq 0 \ \forall j \in V', l \in V' \cup \{0\}, l \neq j, \omega \in \Omega \tag{92}$$

B Additional capacity constraints

Capacity constraints: affine combination predecessors

$$q_j^{\omega} \geq \sum_{i \in V' \cup \{0\}} (qx)_{ij}^{\omega} + d_j \omega \ \forall j \in V', \omega \in \Omega \tag{93}$$

$$(qx)_{ij}^{\omega} \geq d_i^{\omega} x_{ij}^{\omega} \ \forall i \in V' \cup \{0\}, j \in V', \omega \in \Omega \tag{94}$$

$$(qx)_{ij}^{\omega} \geq q_i^{\omega} - Q(1 - x_{ij}^{\omega}) \ \forall i \in V' \cup \{0\}, j \in V', \omega \in \Omega \tag{95}$$

$$d_i^{\omega} \leq q_i^{\omega} \leq Q \ \forall i \in V', \omega \in \Omega \tag{96}$$

$$q_i^{\omega} \geq 0 \ \forall i \in V', \omega \in \Omega \tag{97}$$

In which $(qx)_{ij}^{\omega}$ is a single variable $\forall i \in V' \cup \{0\}, j \in V', \omega \in \Omega$.

C Complexity separation precedence inequalities when using MTZ time constraints

The problem of finding the most violated precedence inequality with certainty, if one exists, is NP-hard for a general optimal solution to the general flow formulation in which the integrality constraints are omitted and time is modeled with the MTZ-inequalities. We will prove this result with a polynomial-time reduction from the decision version of TSP, which is known to be NP-complete.

The outline is as follows: based on any given TSP with at least four cities and strictly positive travel times, we will define an instance of the TWAVRP. Then, we will define a set of optimal solutions to the LP relaxation of the general flow formulation in which time is modeled by the MTZ-inequalities. Next, we will define a set of instances of the corresponding separation problems, $I(i,j)$, in which i and j are parameters. Finally we show that the TSP allows for an objective $\leq \alpha$ if and only if $\min_{ij} \{v(I(i,j))\} < 1$, in which $v(.)$ indicates the objective value of $I(i,j)$.

Consider a TSP with $n \geq 4$ cities and strict positive distances between all cities. We would like to answer the question "does there exist a feasible solution to the TSP with objective $\leq \alpha$". First, we will turn this TSP into an instance of the TWAVRP. Let V' denote the set of nodes corresponding to the cities. Add two depot nodes 0 and $n+1$ and define $V = V' \cup \{0, n+1\}$.

Naturally we define t_{kl} to be the distance between cities k and l, $\forall k, l \in V'$. The distances from and to the depot nodes can be chosen arbitrarily.

For this instance, we pick the exogenous time windows as large as possible. We can do so by setting $s_k = 0$ and $e_k = \infty$ $\forall k \in V' \cup \{0\}$. The endogenous time window width is set to $w_k = \frac{\alpha}{2}$ $\forall k \in V'$.

Demand can be made non restrictive, regardless of the formulation. We define two scenarios $\Omega = \{\omega, \omega'\}$. In both scenarios, demand can be chosen arbitrarily. To make the capacity constraints non restrictive, we pick a large truck capacity of $Q = \sum_{k \in V'} (d_k^{\omega} + d_k^{\omega'})$.

Now we can define the instances of the separation problem, $I(i,j) \forall i, j \in V'$ $(i \neq j)$. These instances have to be feasible for the LP relaxation of the general flow formulation, in which time is modeled by the MTZ-inequalities.

Finally, we set all costs to zero $c_{kl} = 0$ $\forall k, l \in V$. This causes every feasible solution of the LP relaxation to be optimal as well.

Next, we define an optimal solution for each combination of i and j, $i, j \in V'$ $(i \neq j)$. Scenario ω' has an integral solution, in which each customer is served by a single truck, except for i and j which are served by a truck that leaves the depot, visits i, then visits j and then returns to the depot.

Scenario ω has a very fractional solution:

- $x_{0i}^{\omega} = 1$

- $x^{\omega}_{j(n+1)} = 1$
- $x^{\omega}_{kl} = \frac{1}{n-2}\ \forall k, l \in V' \backslash \{i, j\}, k \neq l$
- $x^{\omega}_{il} = \frac{1}{n-2}\ \forall l \in V' \backslash \{i, j\}$
- $x^{\omega}_{kj} = \frac{1}{n-2}\ \forall k \in V' \backslash \{i, j\}$
- all other x-values are equal to 0

This fractional solution is feasible for the LP relaxation of the general flow formulation with MTZ-inequalities for time. The flows are easily verified to be feasible. Note that the MTZ-inequalities (17) use a 'big-M' of $e_k - s_l = \infty$ in our case. This implies that if for any arc the x-variable has a value less than 1, the associated MTZ-inequality becomes useless, thus allowing the instances $I(i, j)$ to be feasible. Because all costs are zero, the described feasible solution is optimal as well.

Now we will answer the following question, for each instance $I(i, j)$: "does instance $I(i, j)$ contain a cut that is violated by ≥ 1". To do so, we will use the notation of the MIP for exact separation in Section 4.2.1.

First, let us ignore constraint (75) in Section 4.2.1. Note that in scenario ω', the only strictly positive weight, selectable arc, is arc (i, j). It is thus optimal to pick $a_j = 1$ and $b_i = 1$, so j is the 'starting node' and i is the 'ending node'.

If we set $s^{\omega}_k = 1$ and $s^{\omega}_l = 1$ for any $k, l \in V', k \neq l$, it is optimal to activate all arcs between k and l by setting $f^{\omega}_{kl} = 1$ and $f^{\omega}_{lk} = 1$ if possible. Because many of the x-variables have the same values, we can state the following: the maximum violation when exactly m nodes from $V' \backslash \{i, j\}$ are activated, is $\frac{1}{n}(m + m^2) - m$, using the number edges in a complete graph. By the convexity of this function in m, it follows that the optimum is found when either $m = 0$ or $m = n$. Clearly, the optimum is 1 for $m = n$.

Now we reconsider constraint (75). The optimal violation of 1 can only be found if no sets of arcs are selected (f-variables) such that a cycle i to j (in scenario ω') to all other nodes to j (in scenario ω) is shorter than $w_i + w_j = \alpha$.

In other words: $v(I(i, j)) < 1$ if and only if the best TSP solution in which j succeeds i has objective $\leq \alpha$. By varying i and j we get the following statement: the TSP has a solution of length $\leq \alpha$ if and only if $\min_{ij} \{v(I(i, j))\} < 1$, which is a polynomial time reduction from TSP to the exact separation problem. This proves that finding the most violated precedence inequality, given an optimal solution to the general flow formulation in which the integrality constraints are omitted and time is modeled by the MTZ-inequalities, is NP-hard. $\square$

D CPLEX 12.5 parameter settings

Parameter	**Setting**	**Effect**
IloCplex::Threads	1	Disable parallel optimization
IloCplex::MIPSearch	1	Use traditional branch-and-cut search
IloCplex::EpGap	0.000001	Set relative mipgap tolerance
IloCplex::TiLim	3600	Stop after one hour of computation
IloCplex::Cliques	-1	Do not generate clique cuts
IloCplex::Covers	-1	Do not generate cover cuts
IloCplex::DisjCuts	-1	Do not generate disjunctive cuts
IloCplex::FlowCovers	-1	Do not generate flow cover cuts
IloCplex::FlowPaths	-1	Do not generate flow path cuts
IloCplex::FracCuts	-1	Do not generate Gomory fractional cuts
IloCplex::GUBCovers	-1	Do not generate GUB cuts
IloCplex::ImplBd	-1	Do not generate implied bound cuts
IloCplex::MIRCuts	-1	Do not generate MIR cuts
IloCplex::MCFCuts	-1	Do not generate MCF cuts
IloCplex::ZeroHalfCuts	-1	Do not generate zero-half cuts
IloCplex::NumericalEmphasis	1	Exercise extreme caution in computation (only used when reporting LP bounds)
LazyConstraintCallback	Add empty	Allows for optimality cuts (only for sub-optimal-path inequalities)

E Tables

The next pages contain additional tables.

Inst.	No capacity constraints						
	None	MTZ	AC pred.	AC succ.	AC both	CH	ICH
1	8.4800	9.5288	9.5957	9.5530	9.8945	9.6073	9.7858
2	10.0600	10.2764	10.2910	10.2549	10.3056	10.2910	11.0131
3	11.4600	12.4053	12.4905	12.4907	12.7504	12.5614	12.9334
4	13.1200	13.5025	13.5510	13.4974	13.5817	13.5510	13.8317
5	10.6600	10.8752	10.9098	10.9102	10.9102	10.9098	11.6665
6	13.7900	14.3349	14.3084	14.2247	14.3084	14.3381	15.1500
7	9.0700	9.5350	9.5270	9.5225	9.6265	9.5592	9.7605
8	13.1100	14.2837	14.4581	14.2215	14.5079	14.4581	15.0210
9	10.9400	11.8349	11.9914	12.0043	12.1223	12.0003	12.9067
10	9.1200	9.3889	9.3914	9.3953	9.4253	9.3995	10.9643
11	9.2300	9.8071	9.8206	9.8375	10.0335	9.8501	10.5557
12	15.0200	15.2331	15.2258	15.2181	15.2623	15.2370	16.7632
13	15.8400	16.7570	16.8411	16.7450	16.9877	16.8499	18.9959
14	11.4900	11.8484	11.8845	11.9092	11.9903	11.8887	12.5565
15	14.4800	14.6090	14.6179	14.6150	14.6457	14.6179	16.0284
16	13.2400	13.3975	13.3941	13.3965	13.4210	13.4010	14.2779
17	12.1000	12.4756	12.4971	12.4760	12.5199	12.4971	13.1449
18	14.1900	14.3071	14.3021	14.2874	14.3047	14.3087	15.3842
19	15.4000	15.8562	15.9754	16.0036	16.0272	15.9754	17.8713
20	15.1900	15.3091	15.3084	15.3129	15.3733	15.3104	16.1098
21	15.8200	16.1804	16.2023	16.1822	16.2513	16.2143	17.2864
22	17.9400	18.7033	18.9014	18.8728	18.9654	18.9014	19.5172
23	15.6600	15.8319	15.8523	15.8642	15.8686	15.8548	18.5607
24	16.3400	16.4987	16.5155	16.5146	16.5384	16.5157	17.5003
25	15.3900	16.0251	16.0213	15.9770	16.0420	16.0520	18.2292
26	18.0400	18.4541	18.4640	18.4820	18.5898	18.4731	20.8095
27	14.4300	14.7727	14.7896	14.7469	14.7904	14.7897	15.1070
28	15.4700	15.6208	15.6255	15.6160	15.6401	15.6300	17.1373
29	15.6400	15.8519	15.8595	15.8520	15.8755	15.8675	16.3578
30	16.9300	17.1089	17.0997	17.0879	17.1109	17.1169	17.9222
31	15.3900	16.0762	16.0669	16.0981	16.4062	16.0880	17.6881
32	16.4600	16.7092	16.7087	16.6992	16.7497	16.7177	18.0832
33	18.0800	18.5184	18.5425	18.5418	18.6249	18.5532	20.3222
34	18.2900	18.4266	18.4205	18.4223	18.4491	18.4301	20.1627
35	14.9100	14.9452	14.9525	14.9523	14.9545	14.9526	15.8546
36	17.9300	18.1697	18.2003	18.2379	18.2551	18.2031	19.2876
37	15.0700	15.3436	15.3467	15.3267	15.3586	15.3553	16.5253
38	15.0600	15.4291	15.4524	15.4931	15.5513	15.4612	18.1776
39	14.2500	14.6201	14.6590	14.6456	14.6970	14.6611	15.9824
40	15.6700	15.7226	15.7249	15.7245	15.7410	15.7268	17.0238

Table 16: Objective for the LP relaxation of the general flow formulation, for various time constraints and no capacity constraints, instances 1-40

	MTZ (capacity)						
Inst.	None	MTZ	AC pred.	AC succ.	AC both	CH	ICH
1	10.8882	10.8882	10.8882	10.8882	10.8902	10.8882	11.3064
2	10.3902	10.3925	10.3933	10.3902	10.3936	10.3933	11.0131
3	12.5343	12.5583	12.5965	12.5935	12.7804	12.6342	13.0889
4	14.0608	14.0608	14.0608	14.0608	14.0608	14.0608	14.3099
5	11.1890	11.1890	11.1890	11.1890	11.1890	11.1890	11.6941
6	14.3269	14.3649	14.3587	14.3312	14.3593	14.3785	15.2778
7	10.3267	10.3267	10.3267	10.3267	10.3267	10.3267	10.8237
8	15.2517	15.2517	15.2517	15.2517	15.2517	15.2517	16.0458
9	12.8712	12.8778	12.8802	12.8743	12.8802	12.8802	13.5733
10	9.5333	9.5377	9.5386	9.5353	9.5419	9.5386	11.0328
11	10.3719	10.3838	10.3941	10.3941	10.4764	10.3973	11.1330
12	15.3069	15.3082	15.3113	15.3088	15.3113	15.3113	16.7889
13	17.2066	17.2126	17.2078	17.2066	17.2764	17.2132	19.4363
14	13.2220	13.2220	13.2220	13.2220	13.2220	13.2220	13.5967
15	14.7878	14.7878	14.7878	14.7878	14.7890	14.7878	16.1024
16	13.6125	13.6125	13.6125	13.6125	13.6125	13.6125	14.3423
17	12.9525	12.9525	12.9525	12.9525	12.9525	12.9525	13.5968
18	14.3961	14.3985	14.3964	14.3961	14.3964	14.3985	15.4747
19	16.0704	16.0737	16.1036	16.1270	16.1270	16.1036	18.2363
20	15.4324	15.4324	15.4324	15.4324	15.4399	15.4324	16.2417
21	16.9306	16.9306	16.9306	16.9306	16.9306	16.9306	17.6963
22	19.8103	19.8103	19.8103	19.8103	19.8103	19.8103	20.3821
23	16.0935	16.0935	16.0935	16.0935	16.0935	16.0935	18.6760
24	16.8110	16.8110	16.8110	16.8110	16.8110	16.8110	17.6459
25	16.6587	16.6587	16.6599	16.6587	16.6637	16.6599	18.5748
26	18.9565	18.9565	18.9565	18.9565	18.9565	18.9565	21.1227
27	15.3044	15.3044	15.3044	15.3044	15.3044	15.3044	15.7933
28	16.1927	16.1927	16.1927	16.1927	16.1927	16.1927	17.2790
29	16.3388	16.3388	16.3388	16.3388	16.3388	16.3388	16.8033
30	17.4866	17.4866	17.4866	17.4866	17.4873	17.4866	18.2617
31	17.2827	17.2827	17.2827	17.2827	17.2859	17.2827	18.4856
32	17.3098	17.3098	17.3098	17.3098	17.3098	17.3098	18.4310
33	19.3881	19.3881	19.3907	19.3907	19.4124	19.3907	20.8768
34	18.8639	18.8639	18.8639	18.8639	18.8639	18.8639	20.4680
35	15.0673	15.0673	15.0673	15.0673	15.0673	15.0673	15.8900
36	18.5848	18.5848	18.5848	18.5848	18.5848	18.5848	19.5293
37	15.9524	15.9524	15.9524	15.9524	15.9524	15.9524	17.1223
38	16.8416	16.8416	16.8416	16.8416	16.8416	16.8416	19.5477
39	16.3758	16.3783	16.3806	16.3778	16.3821	16.3806	17.5591
40	15.8464	15.8464	15.8464	15.8464	15.8464	15.8464	17.0864

Table 17: Objective for the LP relaxation of the general flow formulation, for various time constraints and MTZ capacity constraints, instances 1-40

	2CF						
Inst.	None	MTZ	AC pred.	AC succ.	AC both	CH	ICH
1	12.5667	12.5955	12.5667	12.5667	12.6005	12.5955	12.5667
2	12.6944	12.6944	12.6944	12.6944	12.6944	12.6944	12.6944
3	13.9121	13.9232	13.9121	13.9121	13.9128	13.9232	13.9121
4	15.1504	15.1504	15.1527	15.1515	15.1527	15.1527	15.1515
5	13.2729	13.2729	13.2729	13.2729	13.2729	13.2729	13.2729
6	16.8022	16.8022	16.8022	16.8022	16.8022	16.8022	16.8022
7	11.9782	11.9782	11.9782	11.9782	11.9782	11.9782	11.9782
8	18.3313	18.3313	18.3313	18.3313	18.3313	18.3313	18.3313
9	16.2204	16.2394	16.2463	16.2293	16.2463	16.2463	16.2293
10	13.9905	14.0026	14.0024	13.9947	14.0024	14.0026	13.9947
11	13.9703	13.9703	13.9703	13.9703	13.9703	13.9703	13.9703
12	21.2695	21.2695	21.2695	21.2695	21.2695	21.2695	21.2695
13	24.6412	24.6429	24.6412	24.6412	24.6412	24.6429	24.6412
14	17.1932	17.1932	17.1932	17.1932	17.1932	17.1932	17.1932
15	20.1415	20.1415	20.1415	20.1415	20.1415	20.1415	20.1415
16	18.2286	18.2286	18.2286	18.2286	18.2286	18.2286	18.2286
17	17.7015	17.7015	17.7015	17.7015	17.7015	17.7015	17.7015
18	18.5841	18.5841	18.5841	18.5841	18.5841	18.5841	18.5841
19	23.3988	23.4009	23.4034	23.4028	23.4037	23.4034	23.4028
20	19.1005	19.1005	19.1005	19.1005	19.1005	19.1005	19.1005
21	24.1658	24.1658	24.1658	24.1658	24.1658	24.1658	24.1658
22	24.6123	24.6123	24.6123	24.6123	24.6123	24.6123	24.6123
23	24.9950	24.9950	24.9950	24.9950	24.9950	24.9950	24.9950
24	20.6391	20.6391	20.6391	20.6391	20.6391	20.6391	20.6391
25	25.6145	25.6145	25.6166	25.6145	25.6179	25.6166	25.6145
26	25.6296	25.6296	25.6296	25.6296	25.6296	25.6296	25.6296
27	21.2204	21.2204	21.2204	21.2204	21.2204	21.2204	21.2204
28	22.8260	22.8260	22.8260	22.8260	22.8260	22.8260	22.8260
29	21.2340	21.2340	21.2340	21.2340	21.2340	21.2340	21.2340
30	22.1863	22.1863	22.1863	22.1863	22.1863	22.1863	22.1863
31	25.3040	25.3040	25.3040	25.3040	25.3040	25.3040	25.3040
32	25.5265	25.5265	25.5265	25.5265	25.5265	25.5265	25.5265
33	28.6516	28.6516	28.6516	28.6516	28.6724	28.6516	28.6516
34	29.3650	29.3650	29.3650	29.3650	29.3650	29.3650	29.3650
35	24.2436	24.2436	24.2436	24.2436	24.2436	24.2436	24.2436
36	25.7502	25.7502	25.7502	25.7502	25.7502	25.7502	25.7502
37	23.3144	23.3144	23.3144	23.3144	23.3144	23.3144	23.3144
38	29.5874	29.5874	29.5874	29.5874	29.5874	29.5874	29.5874
39	26.4926	26.4947	26.4952	26.4926	26.4978	26.4952	26.4926
40	26.2951	26.2951	26.2951	26.2951	26.2952	26.2951	26.2951

Table 18: Objective for the LP relaxation of the general flow formulation, for various time constraints and 2CF capacity constraints, instances 1-40

	ICH						
Inst.	None	MTZ	AC pred.	AC succ.	AC both	CH	ICH
1	12.8520	12.8690	12.8952	12.8520	12.9280	12.9067	12.8520
2	12.7587	12.7587	12.7587	12.7587	12.7587	12.7587	12.7587
3	13.9872	13.9983	13.9872	13.9872	13.9880	13.9983	13.9872
4	15.3298	15.3298	15.3298	15.3298	15.3298	15.3298	15.3298
5	13.2898	13.2898	13.2898	13.2898	13.2898	13.2898	13.2898
6	16.8623	16.8623	16.8623	16.8623	16.8623	16.8623	16.8623
7	12.2351	12.2351	12.2351	12.2351	12.2351	12.2351	12.2351
8	18.4673	18.4673	18.4673	18.4673	18.4673	18.4673	18.4673
9	16.3232	16.3311	16.3342	16.3277	16.3342	16.3342	16.3277
10	14.0506	14.0547	14.0546	14.0506	14.0546	14.0547	14.0506
11	14.1364	14.1364	14.1364	14.1364	14.1364	14.1364	14.1364
12	21.4871	21.4871	21.4871	21.4871	21.4871	21.4871	21.4871
13	24.7080	24.7097	24.7080	24.7080	24.7080	24.7097	24.7080
14	17.4216	17.4216	17.4216	17.4216	17.4216	17.4216	17.4216
15	20.3070	20.3070	20.3070	20.3070	20.3070	20.3070	20.3070
16	18.3527	18.3527	18.3527	18.3527	18.3527	18.3527	18.3527
17	17.8123	17.8123	17.8123	17.8123	17.8123	17.8123	17.8123
18	18.7081	18.7081	18.7081	18.7081	18.7081	18.7081	18.7081
19	23.4552	23.4573	23.4598	23.4601	23.4605	23.4598	23.4601
20	19.2914	19.2914	19.2914	19.2914	19.2914	19.2914	19.2914
21	24.4120	24.4120	24.4120	24.4120	24.4120	24.4120	24.4120
22	24.9648	24.9648	24.9648	24.9648	24.9648	24.9648	24.9648
23	25.1704	25.1704	25.1704	25.1704	25.1704	25.1704	25.1704
24	20.8698	20.8698	20.8698	20.8698	20.8698	20.8698	20.8698
25	25.8809	25.8809	25.8809	25.8809	25.8809	25.8809	25.8809
26	25.7725	25.7725	25.7725	25.7725	25.7725	25.7725	25.7725
27	21.5033	21.5033	21.5033	21.5033	21.5033	21.5033	21.5033
28	22.9417	22.9417	22.9417	22.9417	22.9417	22.9417	22.9417
29	21.5805	21.5805	21.5805	21.5805	21.5805	21.5805	21.5805
30	22.4773	22.4773	22.4773	22.4773	22.4773	22.4773	22.4773
31	25.7765	25.7765	25.7765	25.7765	25.7765	25.7765	25.7765
32	25.8228	25.8228	25.8228	25.8228	25.8228	25.8228	25.8228
33	28.8654	28.8654	28.8682	28.8666	28.8731	28.8682	28.8666
34	29.5215	29.5215	29.5215	29.5215	29.5215	29.5215	29.5215
35	24.5038	24.5038	24.5038	24.5038	24.5038	24.5038	24.5038
36	26.1470	26.1470	26.1470	26.1470	26.1470	26.1470	26.1470
37	23.5087	23.5087	23.5087	23.5087	23.5087	23.5087	23.5087
38	29.9059	29.9059	29.9059	29.9059	29.9059	29.9059	29.9059
39	26.7758	26.7768	26.7794	26.7758	26.7794	26.7794	26.7758
40	26.6217	26.6217	26.6217	26.6217	26.6217	26.6217	26.6217

Table 19: Objective for the LP relaxation of the general flow formulation, for various time constraints and ICH capacity constraints, instances 1-40

	No capacity constraints						
Inst.	None	MTZ	AC pred.	AC succ.	AC both	CH	ICH
1	78	0	0	16	15	16	0
2	0	0	0	0	15	15	0
3	0	0	0	0	15	0	0
4	0	0	0	15	31	16	0
5	0	0	0	0	16	16	15
6	0	0	16	0	15	16	0
7	0	0	15	0	15	0	16
8	0	16	0	15	16	31	0
9	0	0	0	0	16	15	0
10	0	0	0	0	15	16	0
11	0	0	0	15	31	31	32
12	0	0	15	16	32	31	31
13	0	0	16	16	31	31	46
14	0	0	15	16	32	31	15
15	0	0	15	16	31	15	31
16	0	0	15	16	31	32	31
17	0	0	0	15	31	31	31
18	0	15	0	31	15	16	31
19	0	0	16	16	32	31	47
20	0	0	0	31	31	31	31
21	0	0	16	47	63	47	109
22	0	0	15	31	78	62	109
23	0	0	16	31	47	31	78
24	0	0	0	47	78	47	62
25	0	0	0	31	47	62	94
26	15	0	16	47	62	62	124
27	0	16	16	47	47	31	63
28	0	0	15	47	62	47	63
29	0	0	15	46	63	47	94
30	0	0	16	47	62	47	62
31	0	0	15	78	172	94	141
32	0	0	15	62	109	78	249
33	0	16	15	94	140	94	171
34	0	16	16	94	94	78	219
35	0	0	15	78	94	62	188
36	0	16	16	94	94	78	250
37	0	0	31	62	94	78	156
38	0	0	16	78	78	94	140
39	0	15	31	78	141	109	140
40	0	0	16	63	78	62	250

Table 20: Runtime in ms for the LP relaxation of the general flow formulation, for various time constraints and no capacity constraints, instances 1-40

	MTZ (capacity)						
Inst.	None	MTZ	AC pred.	AC succ.	AC both	CH	ICH
1	0	15	0	0	16	15	0
2	0	0	15	0	16	16	0
3	0	0	0	16	16	15	16
4	0	0	15	16	15	16	16
5	0	0	16	0	16	0	16
6	0	15	0	0	16	16	16
7	0	0	0	16	16	0	15
8	0	0	0	16	0	15	16
9	0	0	0	15	15	0	31
10	0	0	0	0	16	16	16
11	0	0	0	15	32	31	16
12	0	0	16	15	16	31	31
13	0	0	15	31	32	32	47
14	0	16	16	15	31	31	31
15	0	0	16	16	31	31	47
16	0	16	16	16	31	16	31
17	0	16	0	31	31	31	16
18	15	0	0	16	31	32	32
19	0	0	15	31	63	32	63
20	0	0	0	15	32	16	47
21	0	0	16	62	78	46	219
22	0	0	0	62	62	47	156
23	0	15	15	47	47	32	124
24	0	0	16	47	78	31	109
25	16	0	16	47	63	46	188
26	0	16	0	47	62	47	219
27	15	0	0	47	63	32	141
28	16	0	16	47	46	47	93
29	0	0	16	47	78	47	172
30	16	0	15	62	62	31	109
31	0	16	31	125	125	78	328
32	0	16	15	94	78	78	312
33	0	0	31	124	125	94	515
34	0	15	16	124	78	78	437
35	0	15	15	94	78	63	390
36	0	16	16	141	110	78	405
37	0	16	15	94	93	78	250
38	16	0	15	62	63	62	234
39	0	16	15	93	109	78	390
40	16	16	16	63	78	78	250

Table 21: Runtime in ms for the LP relaxation of the general flow formulation, for various time constraints and MTZ capacity constraints, instances 1-40

			2CF				
Inst.	None	MTZ	AC pred.	AC succ.	AC both	CH	ICH
1	0	0	16	0	31	31	0
2	16	16	16	0	16	15	15
3	16	0	16	15	15	32	16
4	0	0	16	15	32	16	16
5	0	16	15	16	31	15	15
6	0	0	0	0	31	16	16
7	0	0	0	15	15	16	16
8	0	15	16	15	15	32	16
9	16	0	0	0	16	15	16
10	0	0	16	15	15	31	15
11	15	16	16	31	47	46	32
12	15	16	31	47	47	78	47
13	15	16	31	31	63	63	31
14	15	16	31	31	47	62	31
15	15	16	31	32	78	78	31
16	15	16	31	47	62	63	32
17	16	15	31	47	47	62	31
18	16	15	31	31	47	63	46
19	16	16	16	31	63	47	62
20	0	16	31	31	78	78	32
21	32	31	62	78	156	140	110
22	16	16	78	110	140	125	125
23	15	32	62	63	109	109	78
24	32	31	63	109	140	109	109
25	31	31	47	109	219	124	109
26	16	31	78	93	140	110	125
27	31	32	62	94	125	93	109
28	15	31	63	109	125	109	125
29	16	32	78	94	125	124	110
30	31	32	62	93	156	172	125
31	47	47	109	219	343	390	297
32	46	62	109	172	296	203	218
33	47	47	109	234	328	328	266
34	47	47	125	187	266	281	265
35	47	47	140	202	296	234	234
36	47	47	141	219	281	390	281
37	31	47	109	156	234	234	172
38	31	47	140	188	234	187	219
39	47	47	94	156	265	202	218
40	47	46	109	188	249	281	218

Table 22: Runtime in ms for the LP relaxation of the general flow formulation, for various time constraints and 2CF capacity constraints, instances 1-40

				ICH			
Inst.	None	MTZ	AC pred.	AC succ.	AC both	CH	ICH
1	16	0	15	16	15	16	16
2	0	16	15	16	16	31	16
3	0	16	0	16	16	15	15
4	16	0	16	16	16	16	31
5	0	0	16	15	31	32	16
6	16	0	0	16	16	16	15
7	16	16	0	0	16	15	16
8	0	16	16	16	16	15	0
9	0	15	0	15	15	15	15
10	0	0	15	16	16	32	16
11	16	15	16	31	62	62	31
12	15	16	15	46	62	63	78
13	16	16	16	31	47	63	31
14	16	31	31	31	47	62	47
15	15	16	31	47	78	47	47
16	16	16	31	47	47	63	31
17	15	16	31	47	62	47	32
18	16	16	31	47	78	63	47
19	16	15	31	31	62	46	78
20	15	16	31	47	62	47	62
21	46	47	78	94	140	141	141
22	31	47	78	124	202	203	171
23	31	31	62	78	109	124	78
24	46	63	78	125	172	187	203
25	47	63	62	125	125	109	172
26	31	47	62	109	171	125	172
27	47	47	62	125	156	124	124
28	31	47	63	109	125	109	141
29	47	47	78	109	188	124	156
30	32	46	78	93	171	125	140
31	62	109	125	234	296	328	188
32	78	109	156	172	312	343	187
33	78	94	141	202	296	344	203
34	93	109	156	234	296	359	218
35	78	125	171	280	343	343	234
36	93	124	156	203	327	375	203
37	78	93	124	218	296	250	219
38	78	109	125	187	265	234	234
39	94	93	125	219	281	265	202
40	78	94	141	172	234	328	171

Table 23: Runtime in ms for the LP relaxation of the general flow formulation, for various time constraints and ICH capacity constraints, instances 1-40