A Time Bucket Formulation for the TSP with Time Windows

Sanjeeb Dash, Oktay Günlük
IBM Research

Andrea Lodi, Andrea Tramontani
University of Bologna

November 10, 2009

Abstract

The Traveling Salesman Problem with Time Windows (TSPTW) is the problem of finding a minimum-cost path visiting a set of cities exactly once, where each city must be visited within a given time window.

We present an extended formulation for the problem based on partitioning the time windows into sub-windows, which we call *buckets*. We present cutting planes for this formulation that are computationally more effective than the ones known in the literature as they exploit the division of the time windows into buckets. To obtain a good partition of the time windows, we propose an iterative LP-based procedure that may produce buckets of different sizes. The LP relaxation of this formulation yields strong lower bounds for the TSPTW and provides a good starting point for our branch-and-cut algorithm.

We also present encouraging computational results on hard test problems from the literature, namely asymmetric instances arising from a practical scheduling application, as well as randomly generated symmetric instances. In particular, we solve a number of previously unsolved benchmark instances.

1 Introduction

The *Traveling Salesman Problem with Time Windows* (TSPTW) is the problem of finding a minimum-cost path visiting a set of cities exactly once, where each city must be visited within a given time window. The problem is NP-hard as it generalizes the classical *Traveling Salesman Problem* (TSP) and Savelsbergh [27] showed that even finding a feasible solution of TSPTW is NP-complete.

The TSPTW can be found in a variety of important real-life applications such as routing, scheduling, and manufacturing-and-delivery problems. Cook [14] informs us that many companies that inquire about using `Concorde` [13], the TSP solver, are interested in time constraints. Some recent surveys on time-constrained routing problems are given by Desrosiers et al. [16] and Cordeau et al. [15].

Literature review. The first computational approaches for solving the TSPTW, dating back to the nineteen eighties, were given by Christofides, Mingozzi and Toth [12] and Baker [8] and considered

a variant of the problem where the total schedule time has to be minimized. Both papers presented branch-and-bound schemes where the former used a so-called state-space-relaxation approach, whereas the latter exploited a time-constrained critical-path formulation.

In the nineties, several contributions were proposed. Langevin et al. [20] addressed the problem by using a two-commodity flow formulation within a branch-and-bound scheme. Dumas et al. [17] proposed a dynamic-programming approach with sophisticated elimination tests to reduce the state space, while Mingozzi, Bianco and Ricciardelli [23] presented a dynamic-programming algorithm with a generalization of the state-space-relaxation technique which can also be applied to TSPTW problems with precedence constraints. Also in the nineteen nineties, the problem started to receive a lot of attention from the *Constraint Programming* (CP) community. Indeed, the fact that the TSPTW has both routing and scheduling characteristics makes it suitable for CP techniques. Small TSPTW instances were solved as subproblems of a large task assignment problem by Caseau and Koppstein [11]. Later, the TSPTW was the main focus of a paper by Pesant et al. [24] where the authors solved it by enriching a simple CP model with redundant constraints. A variant of the TSPTW, called TSP with *multiple* Time Windows, was solved by the same authors (Pesant et al. [25]) with basically the same algorithmic approach (and a slightly adapted model), thus showing the flexibility of the CP paradigm.

More recently, additional approaches have been considered. Balas and Simonetti [10] proposed a special dynamic-programming approach: under the assumption that (for some initial ordering of the cities) city i precedes city j if $j \geq i + k$ ($k > 0$), the authors described and experimented with an algorithm that is linear in n and exponential in k. For the problems satisfying these conditions, the dynamic-programming procedure finds an optimal solution, while in other cases it can be used as a linear time heuristic to explore an exponential-size neighborhood exactly. Ascheuer, Fischetti and Grötschel [7] considered several formulations for the *asymmetric* version (ATSPTW) of the problem (including a new one introduced in the companion paper, Ascheuer, Fischetti and Grötschel [6]), and computationally compared them within a branch-and-cut scheme. They incorporated up-to-date techniques tailored for the ATSPTW such as data preprocessing, primal heuristics, local search, and variable fixing (in addition to problem-specific separation algorithms). Finally, Focacci, Lodi and Milano [18] proposed a hybrid algorithm merging classical Operations Research techniques (such as reduced-cost fixing, cutting planes and Lagrangean relaxations) for coping with the optimization aspect (the routing part), and CP propagation algorithms for the feasibility aspect (the scheduling part).

Contribution of the paper. We present an extended formulation for the TSPTW based on a partition of time windows into sub-windows which yields a strong LP relaxation, in the sense that the optimal relaxation value yields strong lower bounds on the optimal value. Further, we derive valid inequalities for this formulation, and present efficient heuristics to separate them. Finally, we embed these inequalities in a branch-and-cut algorithm, and test the algorithm on hard problems from the literature, namely asymmetric instances, arising from a practical scheduling application, as well as symmetric randomly generated instances. Our results show that the partition of time

windows into sub-windows is effective in practice for tackling the TSPTW. We are able to solve several previously unsolved benchmark instances.

A variety of time-window discretization methods have been in use for a while in solving routing and scheduling problems with time constraints. Wang and Regan [31, 32] recently proposed a similar formulation for obtaining lower bounds for a vehicle routing problem, and also for the TSPTW. However, the present paper goes further by partitioning the time windows iteratively by solving LPs. The resulting sub-windows are then exploited to devise strong cutting planes which are embedded in a branch-and-cut framework with special emphasis on computational performance.

The paper is organized as follows. In Section 2 we formally state the problem and present the new extended formulation for it. Section 3 presents some valid inequalities for the proposed formulation and discusses their connections with other inequalities which are typically used to tackle TSPTW. Section 4 describes the separation procedures for the proposed inequalities. Section 5 gives the main ingredients of the branch-and-cut algorithm and proposes a procedure to iteratively partition the time windows into sub-windows. Section 6 reports computational results on a set of asymmetric and symmetric benchmark instances, comparing the proposed approach with other effective methods from the literature. Finally, some conclusions are drawn in Section 7.

2 Problem Definition and Formulations

We now formally define the ATSPTW, the directed version of the TSPTW. Let $G = (V, A)$, be a directed graph where each node $i \in V$ represents a city with an associated time window $W_i = [R_i, D_i]$. We call R_i the *release time* and D_i the *deadline* of node i. Each arc $(i, j) \in A$ has an associated travel cost $c_{ij} \geq 0$, and a travel time $\theta_{ij} > 0$. We assume that all data is integral. Moreover, there are two special nodes $p, q \in V$. What we call a *tour* is a permutation of V that starts with node p and ends with node q. Given a tour, we associate an *arrival* time and a *start* time with each node as follows. The arrival and start times at p are its release time R_p. For any other node j, the arrival time is the start time at the previous node (say i) in the tour plus θ_{ij} (therefore, the processing time at each node is 0). The start time at node i is defined as the maximum of the arrival time at i and R_i. A tour is *feasible* if the start time at each node is contained in its time window. An optimal solution to the problem is a feasible tour with minimum cost. Early arrivals are allowed, i.e., a tour can arrive at a node before its release time, in which case the tour "waits" until the release time of the node.

Our definition of the problem requires two special "start" and "end" nodes p and q. In some variants of the TSPTW studied in the literature, any permutation of the nodes is allowed (subject to time window constraints). Such instances can be translated into our setting by creating two artificial nodes p and q, adding arcs with cost zero and travel time 1 from p to all nodes and from all nodes to q, and setting their time windows to be $W_p = [m, m]$ and $W_q = [M, M]$, where m is less than all release times, and M is larger than all deadlines. In other variants of the TSPTW, nodes have processing times. One can incorporate node processing times into travel times between

nodes to obtain the variant we consider here.

Throughout the paper we use $V^+(i)$ to denote the set of nodes j such that there is an arc from i to j, that is, $V^+(i) := \{j \in V : (i,j) \in A\}$. Similarly, $V^-(i) := \{j \in V : (j,i) \in A\}$. We next describe three mixed integer programming formulations for the problem. The first one is the so-called Big-M Formulation [7]. The next one is called the Time Indexed Formulation, and we call the third one the Time Bucket Formulation.

2.1 Big-M Formulation (BMF)

This formulation uses a binary variable x_{ij} for each arc $(i,j) \in A$ to denote whether or not node j follows node i in the tour. In addition, there is a variable s_i associated with each node $i \in V$ to represent its start time in the tour. The *Big-M Formulation* (BMF) is:

$$\min \quad \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{1}$$

$$\sum_{j\in V^+(i)} x_{ij} = 1 \qquad \forall\ i \in V \setminus \{q\}, \tag{2}$$

$$\sum_{k\in V^-(i)} x_{ki} = 1 \qquad \forall\ i \in V \setminus \{p\}, \tag{3}$$

$$s_i + \theta_{ij} - (1 - x_{ij})M_{ij} \le s_j \qquad \forall\ (i,j) \in A, \tag{4}$$

$$R_i \le s_i \le D_i \qquad \forall\ i \in V, \tag{5}$$

$$x_{ij} \in \{0,1\} \qquad \forall\ (i,j) \in A, \tag{6}$$

where $M_{ij} = D_i - R_j + \theta_{ij}$. The constraints (2) and (3) ensure that the x_{ij} variables with value 1 in a feasible integral solution correspond to the union of a path from p to q and a collection of directed cycles. The constraints (4) ensure that start times at the nodes are increasing along any path (as $\theta_{ij} > 0$) and therefore directed cycles cannot exist in the solution. Finally, the constraints (4) and (5) together ensure that the solution respects time windows and defines a feasible tour.

2.2 Time Indexed Formulation (TIF)

Time indexed formulations have been studied for routing and scheduling problems with time constraints starting with the early work of Appelgren [3, 4] and Levin [21]. A time indexed formulation for the ATSPTW with time dependent travel times is given by Albiach, Sanchis and Soler [2]; their formulation reduces to the one discussed here if the travel times are constant. In the formulation below, instead of the start time variables s_i, there are binary variables z_i^t associated with each $t \in W_i$ such that $z_i^t = 1$ if and only if the start time at node i is t. In addition, there are binary variables y_{ij}^t associated with each arc $(i,j) \in A$ and each integral $t \in W_i$ such that $y_{ij}^t = 1$ if and only if the start time at node i is t and arc (i,j) is present in the tour (recall that travel times are integral). We assume that y_{ij}^t is present in the formulation only if $t + \theta_{ij} \le D_j$, i.e., one can start at time t at node i, travel along arc (i,j) and arrive at j by its deadline.

We use $I_k(i,t)$ to denote the collection of possible start times at node k assuming that the start time at node i is t and arc (k,i) is selected:

$$I_k(i,t) \;=\; \{\tau \in W_k : \max\{\tau + \theta_{ki}, R_i\} = t\}.$$

In other words, if the start time t at node i is R_i, then the start time at node k in a feasible tour is some $\tau \in W_k$ satisfying $\tau \leq R_i - \theta_{ki}$, otherwise the start time at node k is exactly $t - \theta_{ki}$ if it belongs to W_k.

The following is the *Time Indexed Formulation* (TIF) for TSPTW:

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in A} c_{ij}x_{ij} & & \\
& \sum_{t\in W_i} z_i^t = 1 & \forall\, i \in V, & (7)\\
& \sum_{j\in V^+(i)} y_{ij}^t = z_i^t & \forall\, i \in V\setminus\{q\}, \forall\, t \in W_i, & (8)\\
\sum_{k\in V^-(i)} & \sum_{\tau\in I_k(i,t)} y_{ki}^\tau = z_i^t & \forall\, i \in V\setminus\{p\}, \forall\, t \in W_i, & (9)\\
& \sum_{t\in W_i} y_{ij}^t = x_{ij} & \forall\, (i,j) \in A, & (10)\\
& x_{ij}, y_{ij}^t, z_i^t \in \{0,1\} & \forall\, i \in V, \forall\, (i,j) \in A, \forall\, t \in W_i. & (11)
\end{aligned}$$

The constraints (7) above assert that a tour must start at each node i within the window W_i. Constraint (8) asserts that a tour with start time t at node $i \neq q$ must leave the node at time t along some arc, whereas constraints (9) assert that for a node $i \neq p$, a tour has a start time at the previous node (say k) contained in $I_k(i,t)$. Clearly any feasible tour can be mapped to an integral solution of TIF by setting $x_{ij} = 1$ if arc (i,j) is used in the tour, and $z_i^t = 1$ if the tour has start time t at node i. Note that if $x_{ij} = 1$, and for some t, t' we have $z_i^t = 1$ and $z_j^{t'} = 1$, then $t < t'$ as $\theta_{ij} > 0$ for all arcs in A. Therefore, there cannot be any directed cycles in the support of the solution and every integral solution of TIF defines a feasible tour. We now prove the following relationship between the LP relaxations of TIF and TBF.

Proposition 1 *The LP relaxation of TIF dominates the LP relaxation of BMF.*

Proof. Let (x,y,z) be a feasible solution of the LP relaxation of TIF. Let $s \in \mathbb{R}^{|V|}$ be defined as

$$s_i = \sum_{t\in W_i} t z_i^t \text{ for all } i \in V.$$

We next show that (x,s) is a feasible solution of the LP relaxation of BMF; the proposition will follow.

Firstly, (2) holds as

$$\sum_{j\in V^+(i)} x_{ij} = \sum_{j\in V^+(i)} \sum_{t\in W_i} y_{ij}^t = \sum_{t\in W_i} \sum_{j\in V^+(i)} y_{ij}^t = \sum_{t\in W_i} z_i^t = 1.$$

The first equality above is implied by (10), the third equality by (8) and the last one by (7). To see that (3) holds, note that

$$\sum_{k\in V^-(i)} x_{ki} = \sum_{k\in V^-(i)} \sum_{t\in W_k} y^t_{ki} = \sum_{k\in V^-(i)} \sum_{t'\in W_i} \sum_{t\in I_k(i,t')} y^t_{ki} = \sum_{t'\in W_i} z^{t'}_i = 1$$

where the last two equalities are implied by equations (8) and (9), respectively.

The constraints (5) are clearly satisfied by s. We now show that the constraints (4) are satisfied by (x, s). Let $\bar{z}^t_i = z^t_i - y^t_{ij}$. Then

$$\sum_{t\in W_i} \bar{z}^t_i = \sum_{t\in W_i} z^t_i - \sum_{t\in W_i} y^t_{ij} = 1 - x_{ij}$$

by equations (7) and (10). Therefore

$$\begin{aligned} s_i = \sum_{t\in W_i} t z^t_i = & \quad \sum_{t\in W_i} t\bar{z}^t_i + \sum_{t\in W_i} t y^t_{ij} \\ \leq & \quad D_i \sum_{t\in W_i} \bar{z}^t_i + \sum_{t\in W_i} t y^t_{ij} = D_i(1 - x_{ij}) + \sum_{t\in W_i} t y^t_{ij}. \end{aligned} \tag{12}$$

Now, using the fact that $M_{ij} = D_i + \theta_{ij} - R_j$, we have

$$s_i + \theta_{ij} - (1 - x_{ij})M_{ij} = s_i + \theta_{ij}x_{ij} - (1 - x_{ij})(D_i - R_j).$$

The inequality in (12) and (10) imply that the last term above is less than or equal to

$$R_j(1 - x_{ij}) + \theta_{ij}x_{ij} + \sum_{t\in W_i} t y^t_{ij} = R_j(1 - x_{ij}) + \sum_{t\in W_i} (t + \theta_{ij}) y^t_{ij}.$$

Similarly, writing $\bar{z}^\tau = z^\tau - \sum_{t\in I_i(j,\tau)} y^t_{ij}$, we have $\sum_{\tau\in W_j} \bar{z}^\tau = 1 - x_{ij}$. Further

$$\begin{aligned} s_j = \sum_{\tau\in W_j} \tau z^\tau_j = \sum_{\tau\in W_j} \tau \bar{z}^\tau + \sum_{\tau\in W_j} \tau \sum_{t\in I_i(j,\tau)} y^t_{ij} \quad & \geq R_j(1 - x_{ij}) + \sum_{\tau\in W_j} \tau \sum_{t\in I_i(j,\tau)} y^t_{ij} \\ & \geq R_j(1 - x_{ij}) + \sum_{t\in W_i} (t + \theta_{ij}) y^t_{ij}. \end{aligned}$$

□

2.3 Time Bucket Relaxation (TBR)

We next present a relaxation of the TSPTW obtained by aggregating some variables in the TIF presented above; some of its feasible solutions may not correspond to feasible tours of the TSPTW. This was noted earlier by Wang and Regan who propose a very similar relaxation (the "under-constrained method") for obtaining lower bounds for a vehicle routing problem in [31], and for the TSPTW in [32]. In Section 2.4, we discuss how to obtain an exact formulation for the TSPTW by adding the so-called *subtour elimination* constraints and *infeasible path* inequalities to this relaxation.

We call this relaxation the *Time Bucket Relaxation* (TBR); unlike the TIF, we do not have variables for each time index. Instead, we partition a node time-window into a collection of non-overlapping intervals and define variables for these intervals. More precisely, the time window W_i for a node $i \in V$ is divided into a set of *buckets* (sub-intervals) $B_i = \{b_1^i, \ldots, b_L^i\}$, where the buckets $b_k^i = [r_{b_k^i}, d_{b_k^i}]$ satisfy (i) $r_{b_1^i} = R_i$, (ii) $d_{b_L^i} = D_i$, and (iii) $r_{b_{k+1}^i} > d_{b_k^i}$ for all k. We allow $W_i \neq \bigcup_{b \in B_i} [r_b, d_b]$ as long as each missing time instant $t \in W_i$ cannot be a valid starting time of a feasible tour at node i. Using earlier notation, a sufficient condition for such a time instant t is $\bigcup_{k \in V^-(i)} I_k(i,t) = \emptyset$.

As before, binary variables x_{ij} indicate whether or not arc $(i,j) \in A$ is a part of the tour. We define binary variables z_i^b associated with each bucket $b \in B_i$ of a node $i \in V$ such that $z_i^b = 1$ if and only if bucket b is selected for node i (b is then called the *starting bucket* at i). We also define binary variables y_{ij}^b associated with each arc $(i,j) \in A$ and each bucket $b \in B_i$ such that $y_{ij}^b = 1$ if and only if $x_{ij} = 1$ and $z_i^b = 1$.

We let $I_k(i,b)$ denote the collection of possible starting buckets at node k assuming that arc (k,i) is selected and the starting bucket at node i is b. More precisely, for a bucket $b_\ell \in B_i$

$$I_k(i, b_\ell) = \{b \in B_k \,:\, d_{b_{\ell-1}} < r_b + \theta_{ki} \leq d_{b_\ell}\}$$

where we assume $d_{b_0} = -\infty$. Conversely, if the starting bucket at node k is b and arc (k,i) is selected, then the starting bucket at node i is denoted by $N_i(k,b)$ and is defined as the bucket $\beta \in B_i$ such that $b \in I_k(i,\beta)$. If $r_b + \theta_{ki} > D_i$, we define $N_i(k,b)$ to be null. As for the TIF, we assume that y_{ki}^b is present in the formulation only if $N_i(k,b)$ is not null. The following is the time bucket relaxation of TSPTW:

$$\begin{aligned}
\min \quad & \sum_{(i,j) \in A} c_{ij} x_{ij} & & \\
& \sum_{b \in B_i} z_i^b = 1 & \forall\, i \in V, & (13)\\
& \sum_{j \in V^+(i)} y_{ij}^b = z_i^b & \forall\, i \in V \setminus \{q\}, \forall\, b \in B_i, & (14)\\
& \sum_{k \in V^-(i)} \sum_{\beta \in I_k(i,b)} y_{ki}^\beta = z_i^b & \forall\, i \in V \setminus \{p\}, \forall\, b \in B_i, & (15)\\
& \sum_{b \in B_i} y_{ij}^b = x_{ij} & \forall\, (i,j) \in A, & (16)\\
& x_{ij}, y_{ij}^b, z_i^b \in \{0,1\} & \forall\, i \in V, \forall\, (i,j) \in A, \forall\, b \in B_i. & (17)
\end{aligned}$$

Note that equation (13) implies that for each node $i \in V$, there is precisely one $b \in B_i$ for which $z_i^b = 1$. Therefore, if $i \neq q$, equation (14) implies that precisely one $y_{ij}^b = 1$ for some $j \in V^+(i)$. Similarly, if $i \neq p$, equation (15) implies that precisely one $y_{ki}^\beta = 1$ for some $k \in V^-(i)$. Therefore, if $i \neq q$, there is precisely one node $j \in V^+(i)$ for which $x_{ij} = 1$ and, if $i \neq p$, there is precisely one node $k \in V^-(i)$ for which $x_{ki} = 1$. Consequently, a feasible solution to the TBR corresponds to the

union of a directed path from p to q and a collection of disjoint cycles. Unlike the TIF, a solution may not correspond to a feasible tour, as disjoint cycles may be present in the solution. To see this, consider a pair of nodes i and j with common time windows of $[0, 10]$, and $\theta_{ij} = \theta_{ji} = 10$, and assume there is only one time bucket at each node (equaling the time windows), namely b (at i) and b' (at j). Then the values $x_{ij} = x_{ji} = z_i^b = z_j^{b'} = 1$ are not ruled out by the TBR constraints and the definition of $I_k(i, b)$.

However, all feasible solutions to the TSPTW correspond to a feasible solution for the TBR. The proof of this claim can be found in Tramontani [30].

Proposition 2 ([30]) *Any feasible tour for TSPTW is also feasible for the TBR and therefore TBR is a relaxation for TSPTW.*

2.4 Time Bucket Formulation (TBF)

The TBR can be turned into a valid formulation for TSPTW with the subtour elimination and infeasible path constraints. For proper subsets S, S' of V, let $\delta(S, S')$ be the set of arcs with their tail in S and head in S', i.e., $\delta(S, S') = \{(i, j) \in A : i \in S, j \in S'\}$. Further, let $\delta(S) = \delta(S, V \setminus S)$ and denote $V \setminus S$ by $\bar{S}$. Clearly, if $q \notin S$, then any feasible tour has at least one arc that connects nodes in S to nodes in $\bar{S}$, i.e., an arc in $\delta(S)$. Therefore, the well known *subtour elimination* constraint (SEC)

$$\sum_{(i,j)\in\delta(S)} x_{ij} \geq 1 \tag{18}$$

is valid for the TSPTW for all $S \subseteq V \setminus \{q\}$. If a feasible solution to the TBR also satisfies all subtour elimination constraints, then it defines a tour.

Let $P = (v_1, v_2, \ldots, v_h)$ be an ordered set of nodes in $S \subseteq V$. We call P an infeasible path if it cannot be contained in any feasible tour. Clearly, any feasible tour satisfies the *infeasible path* constraint

$$\sum_{i=1}^{h-1} x_{v_i,v_{i+1}} \leq h - 2 \tag{19}$$

for every infeasible path P. If a solution to the TBR is a tour which contains no infeasible paths (in particular, if the solution itself is not an infeasible path from p to q) then it defines a feasible solution to the TSPTW. Note that, even though it is NP-hard to separate these inequalities for a fractional solution, it is easy to do so for an integral solution (see Section 4.2 for details).

Therefore, one can formulate the TSPTW using the x variables, and an exponential number of constraints (19), as in Ascheuer, Fischetti and Grötschel [6, 7]. We instead use this observation to obtain a new formulation for the TSPTW which we call the *Time Bucket Formulation* (TBF). The new formulation is obtained by simply adding all possible subtour elimination constraints (18) and all possible infeasible path constraints (19) to the TBR. In practice, these constraints should clearly be used as cutting planes in a branch-and-cut framework.

3 Valid Inequalities

In this section we describe valid inequalities for the TBF. Some of these inequalities are due to Ascheuer, Fischetti and Grötschel [6, 7] and they are defined on x variables only. The remaining inequalities are new and they involve y variables of the TBF. We start by defining a "bucket" graph $G' = (\mathcal{B}, A_{\mathcal{B}})$ where $\mathcal{B} = \bigcup_{i \in V} B_i$ and

$$A_{\mathcal{B}} = \bigcup_{(i,j) \in A} \Big\{ (b, b') \ : \ b \in B_i, b' \in B_j, b' = N_j(i, b) \Big\}.$$

There is a one-to-one mapping from the arcs in $A_{\mathcal{B}}$ to the variables y_{ij}^b of TBF. We let μ denote this mapping, i.e., $\mu : A_{\mathcal{B}} \rightarrow V \times V \times \mathcal{B}$ such that for an arc $(b, b') \in A_{\mathcal{B}}$, $\mu((b, b')) = (i, j, b)$ where b is a bucket of node i, b' is a bucket of node j, and $b' = N_j(i, b)$. Recall that $N_j(i, b)$ is the starting bucket at node j if arc (i, j) is selected and the starting bucket at node i is b. There is also a one-to-one correspondence between the nodes in $\mathcal{B}$ and the variables z_i^b. Therefore, every feasible solution of TBF corresponds to a path in G' that starts from a bucket in B_p and ends at a bucket of B_q while visiting exactly one bucket of each of the remaining nodes.

For $S \subseteq V$, we use $B(S) = \cup_{i \in S} B_i$ to denote the set of buckets associated with the nodes in S. For $B \subseteq \mathcal{B}$, we use $V(B) = \{i \in V : B_i \subseteq B\}$ to denote the set of nodes which have all their buckets contained in B. Further, we define $\bar{B} = \mathcal{B} \setminus B$.

3.1 Bucket Subtour Elimination Constraints

Recall the subtour elimination inequalities (18) that are valid for TSPTW. The inequalities

$$\sum_{(i,j,b) \in \mu(\delta(B))} y_{ij}^b \geq 1, \tag{20}$$

where $B \subseteq \mathcal{B} \setminus B_q$ and $B_t \subseteq B$ for some $t \in V$, involve buckets and generalize inequalities (18).

Proposition 3 *Inequalities (20) are valid for TBF and they subsume inequalities (18).*

Proof. Every feasible solution of TBF corresponds to a path in G' that visits exactly one bucket of each node and ends at a bucket of B_q. Therefore, the path has to use an arc of $\delta(B)$ at least once as $B_q \subseteq \bar{B}$ and $B_t \subseteq B$ for some $t \in V$. Thus the inequalities (20) are valid. To see that these inequalities subsume (18), observe that for any $S \subseteq V \setminus \{q\}$, inequality (20) with $B = \cup_{i \in S} B_i$ implies inequality (18) as $\sum_{(i,j) \in \delta(S)} x_{ij} = \sum_{(i,j) \in \delta(S)} \sum_{b \in B_i} y_{ij}^b = \sum_{(i,j,b) \in \mu(\delta(B))} y_{ij}^b$. □

3.2 Sequential Ordering Polytope (SOP) Inequalities

SOP inequalities are based on a concept of "precedence" between pairs of nodes and were introduced by Balas, Fischetti and Pulleyblank [9] in the context of the precedence-constrained TSP. These inequalities are also effective for TSPTW where precedences between nodes are inferred based on time windows, see [7]. A node $i \in V$ *precedes* node $j \in V \setminus \{i\}$ if j has to be visited after i in any

feasible tour; we denote this as $i \prec j$. We extend this definition to subsets of nodes as follows: for $S, S' \in V$, we say $S \prec S'$ if each node in S precedes every node in S'.

For $S \subseteq V$, let $\pi(S) = \{i \in V : i \prec j \text{ for some } j \in S\}$, and notice that for any feasible tour, if $i \in S \cap \pi(S)$, then i cannot be the last node of S visited by the tour. Similarly, let $\sigma(S) = \{j \in V : i \prec j \text{ for some } i \in S\}$ and note that $j \in S \cap \sigma(S)$ cannot be the first node of S visited by a feasible tour. For any set $S \subseteq V$, let

$$\delta_\pi(S) = \delta(S \setminus \pi(S), \bar{S} \setminus \pi(S)) \text{ and } \delta_\sigma(\bar{S}) = \delta(\bar{S} \setminus \sigma(S), S \setminus \sigma(S)).$$

For all $S \subseteq V \setminus \{p, q\}$, the following π and σ-inequalities are valid for BMF and therefore for TBF:

(π-inequalities)
$$\sum_{(i,j) \in \delta_\pi(S)} x_{ij} \geq 1, \tag{21}$$

(σ-inequalities)
$$\sum_{(i,j) \in \delta_\sigma(\bar{S})} x_{ij} \geq 1. \tag{22}$$

It is known that the constraints (21) and (22) dominate subtour elimination constraints (18) [9].

For any ordered set $P = (v_1, \ldots, v_h)$ of nodes from V, define $\theta(P) = \sum_{i=1}^{h-1} \theta_{v_i, v_{i+1}}$. Then, given disjoint sets of nodes X, Y with $X \prec Y$, let

$$Z = \{k \in V \setminus (X \cup Y) : \exists\, i \in X, j \in Y \text{ with } R_i + \theta(i, k, j) > D_j\}, \tag{23}$$

and note that a path from i to k to j arrives at node j after D_j even if it starts at i at time R_i and cannot be part of a feasible tour. Further, let

$$Q := \{(u, v) \in A : \exists\, i \in X, j \in Y \text{ with } R_i + \theta(i, u, v, j) > D_j\}. \tag{24}$$

Finally, let $W = \pi(X) \cup \sigma(Y) \cup Z$. Then, for any $S \subset V$ such that $X \subseteq S$, $Y \subseteq \bar{S}$, the following (π, σ)-inequalities are also valid for BMF and therefore for TBF:

((π, σ)-inequalities)
$$\sum_{(i,j) \in \delta(S \setminus W, \bar{S} \setminus W) \setminus Q} x_{ij} \geq 1. \tag{25}$$

These inequalities are in fact a strengthened version of the classical (π, σ)-inequalities for the precedence-constrained TSP (see [6, 7]).

3.2.1 Bucket SOP Inequalities

For buckets, we extend the concept of precedence as follows: for a node $i \in V$, we say that bucket $b \in B_i$ precedes node $j \in V$ if all feasible tours that visit node i at bucket b (i.e., $z_i^b = 1$), have to visit node j after node i. We denote this precedence as $b \prec j$. In a similar fashion, we say that a node $j \in V$ precedes a bucket $b \in B_i$ if all feasible tours that visit node i at bucket b, have to visit node j before node i. We denote this precedence as $j \prec b$. Note that the original node precedence relationship implies bucket-node precedences as follows: $i \prec j \Rightarrow b \prec j$ for all $b \in B_i$ and $i \prec b$

for all $b \in B_j$. Furthermore, the bucket-node precedence relationship can be combined with node precedence relationships as follows: for a bucket b and nodes $j, k \in V$, $b \prec j$ and $j \prec k$ implies $b \prec k$, while $j \prec b$ and $k \prec j$ implies $k \prec b$.

Let $B \subseteq \mathcal{B}$ be a collection of buckets. We next abuse notation and use $\pi(B)$ to denote the set of buckets which precede nodes with all buckets contained in B, in other words

$$\pi(B) = \{b \in \mathcal{B} : b \prec i, \text{ where } i \in V \text{ and } B_i \subseteq B\}.$$

Similarly, we define $\sigma(B)$ as

$$\sigma(B) = \{b \in \mathcal{B} : i \prec b, \text{ where } i \in V \text{ and } B_i \subseteq B\}.$$

Let $B \subseteq \mathcal{B}$ such that $B_t \subseteq B$ for some node $t \in V$ and $B_p \cup B_q \subseteq \bar{B}$. We call the following inequalities bucket SOP inequalities:

$$(\pi_B\text{-inequalities }) \qquad \sum_{(i,j,b)\in\mu(\delta_\pi(B))} y_{ij}^b \geq 1, \tag{26}$$

$$(\sigma_B\text{-inequalities }) \qquad \sum_{(i,j,b)\in\mu(\delta_\sigma(\bar{B}))} y_{ij}^b \geq 1. \tag{27}$$

It is easy to see that bucket SOP inequalities (26) and (27) dominate bucket subtour elimination inequalities (20).

Proposition 4 *The Bucket SOP inequalities (26) and (27) are valid for TBF and dominate the SOP inequalities (21) and (22), respectively.*

Proof. As discussed earlier, a feasible solution to TBF corresponds to a directed path in G' that starts with a bucket in B_p, ends at a bucket in B_q and visits exactly one bucket associated with the remaining nodes. As $B_t \subseteq B$ for some $t \in V$ and $B_p \cup B_q \subseteq \bar{B}$ by assumption, the path must start at one of the buckets in $\bar{B}$, must visit one of the buckets in B and must end at one of the buckets in $\bar{B}$. Therefore, the path must use an arc in $\delta(\bar{B})$ and an arc in $\delta(B)$ at least once. Let (b, b') be the last arc on the path that crosses from B to $\bar{B}$ and let $b \in B_i$ and $b' \in B_j$. Notice that b and $b' \notin \pi(B)$. If $b \in \pi(B)$, then there is a node k with $B_k \subseteq B$ such that $b \prec k$, which means that some bucket of B_k is visited after b, a contradiction. The same argument holds for b'. Therefore $(b, b') \in \delta_\pi(B)$ and the inequality (26) is valid. One can similarly argue that if (b, b') is the first arc crossing from $\bar{B}$ to B in a feasible solution to TBF, then $b, b' \notin \sigma(B)$. Therefore inequality (27) is valid.

To see that the π_B-inequalities dominate the π-inequalities, consider the π-inequality (22) with $S \subseteq V \setminus \{p, q\}$. Recall that $B(S) = \cup_{i\in S} B_i$. Define X to be $\delta(B(S) \setminus B(\pi(S)), B(\bar{S}) \setminus B(\pi(S)))$. Now the left-hand side of the π-inequality can be written as

$$\sum_{(i,j)\in\delta_\pi(S)} x_{ij} = \sum_{(i,j)\in\delta_\pi(S)} \sum_{b\in B_i} y_{ij}^b = \sum_{(i,j,b)\in\mu(X)} y_{ij}^b.$$

Let $Y = \delta(B(S) \setminus \pi(B(S)),\, B(\bar{S}) \setminus \pi(B(S)))$; then the π_B-inequality with B replaced by $B(S)$ is

$$\sum_{(i,j,b)\in\mu(Y)} y^b_{ij} \geq 1. \tag{28}$$

As $B(\pi(S)) \subseteq \pi(B(S))$, it follows that Y is contained in X and

$$\sum_{(i,j,b)\in\mu(X)} y^b_{ij} \geq \sum_{(i,j,b)\in\mu(Y)} y^b_{ij}.$$

Therefore the π_B-inequalities dominate the π-inequalities. The proof of domination in the case of σ_B-inequalities is very similar. □

When $B = B(S)$ for some $S \subseteq V$ we call the corresponding bucket SOP inequalities *simple bucket* SOP inequalities, and more specifically, simple π_B-inequalities and simple σ_B-inequalities.

It is easy to extend these ideas to generalize the (π, σ)-inequalities (25) as follows: Let X, Y be two disjoint subsets of nodes with $X \prec Y$. For any $B \subset \mathcal{B}$ such that $X \subseteq V(B)$ and $Y \subseteq V(\bar{B})$, the following inequalities are valid for TBF:

$$((\pi_B, \sigma_B)\text{-inequalities }) \qquad \sum_{(i,j,b)\in\mu(\delta(B\setminus\tilde{W}, \bar{B}\setminus\tilde{W})\setminus\tilde{Q})} y^b_{ij} \geq 1. \tag{29}$$

where $\tilde{W} := \pi(B(X)) \cup \sigma(B(Y)) \cup B(Z)$, $\tilde{Q} = \{(b, b') \in A_{\mathcal{B}} : (i, j) \in Q, b \in B_i, b' \in B_j\}$ and the sets Z and Q are defined as in (23) and (24). Once again, if we let $B = B(S)$ for some S with $X \subseteq S$, and $Y \subseteq \bar{S}$, we call the resulting inequality a simple (π_B, σ_B)-inequality.

3.3 Infeasible Path and Tournament Constraints

Let $P = (v_1, v_2, \ldots, v_h)$, be a directed elementary path in G and let $|P| = h - 1$ denote the number of arcs on P. The path P is called an *infeasible path* if it cannot be contained in any feasible tour. It is known that deciding whether a given path is infeasible or not is NP-complete. However, there are some simple conditions that imply infeasibility. In particular the path P is infeasible if

$$r_{v_1} + \theta(P) > D_{v_h}, \tag{30}$$

where $\theta(P) = \sum_{i=1}^{h-1} \theta_{v_i, v_{i+1}}$ denotes the length of the path. In addition, the path P is infeasible if for some node v_t that is not on the path, both $P' = (v_t, v_1, v_2, \ldots, v_h)$ and $P'' = (v_1, v_2, \ldots, v_h, v_t)$ are infeasible. The infeasibility of the paths P' and P'' can again be checked using (30). As discussed in Section 2.4, given an infeasible path P, the corresponding infeasible path elimination constraint (19) is valid for TBF. Furthermore, these inequalities can be strengthened to obtain the so-called *tournament constraints* (TOURs)

$$\sum_{(i,j)\in T(P)} x_{i,j} \leq |P| - 1, \tag{31}$$

where $T(P) = \{(i, j) \in A : (i, j) = (v_i, v_j) \text{ for some } 1 \leq i < j \leq h\}$. The tournament constraints were originally introduced by Ascheuer, Fischetti and Grötschel [6]. It is known that if P_1 and P_2

are two infeasible paths such that P_1 is contained in P_2, then the tournament constraint associated with P_2 is dominated by the one associated with P_1.

Let $S = \{v_1, \ldots, v_h\}$ be the collection of nodes associated with path $P = (v_1, \ldots, v_h)$ and let $\Psi(S)$ denote the collection of all possible paths that visit all nodes in S. As described in [7], if all paths in $\Psi(S)$ are infeasible then the following stronger inequality

$$\sum_{(i,j)\in A(P)} x_{i,j} \leq |P| - 1, \tag{32}$$

where $A(P) = \{(i,j) \in A : (i,j) = (v_i, v_j) \text{ for some } 1 \leq i, j \leq h\}$, is also valid for BMF and TBF. Whenever a violated tournament constraint (31) is identified it is customary to try to strengthen it this way.

3.3.1 Bucket Tournament Inequalities

We extend inequalities (31) by considering time buckets as follows. Let $P = (v_1, \ldots, v_h)$ be a path and let $b_t \in B_{v_1} = \{b_1, \ldots, b_k\}$ be the earliest bucket of v_1 such that $r_{b_t} + \theta(P) > D_{v_h}$. In other words, if the starting bucket at node v_1 is one of $L = \{b_t, b_{t+1}, \ldots, b_k\}$, then a feasible tour cannot visit the nodes in $\{v_1, \ldots, v_h\}$ in the order specified by P. Note that P itself is not necessarily an infeasible path if $t \neq 1$. It is easy to see that the following inequality

$$\sum_{b\in\{b_t, b_{t+1}, \ldots, b_k\}} z^b_{v_1} + \sum_{(i,j)\in T(P)} x_{i,j} \leq |P| \tag{33}$$

is valid for TBF. Notice that if one of the buckets in L is chosen for node v_1, then P becomes an infeasible path and inequality (33) becomes the tournament constraint (32). Conversely, if one of $\{b_1, \ldots, b_{t-1}\}$ is chosen, the inequality is implied by the subtour elimination constraint.

It is possible to strengthen these inequalities as follows: Let $S = \{v_1, \ldots, v_h\}$ and for $v \in V \setminus S$, let L_v denote the collection of time buckets at node v that cannot be visited if arc (v, v_1) is chosen together with the path P. More precisely, $L_v = \{b \in B_v : r_b + \theta_{v,v_1} + \theta(P) > D_{v_h}\}$. We define the *bucket tournament inequality* (BTOUR) as follows:

$$\sum_{v\in V\setminus S} \sum_{b\in L_v} y^b_{v,v_1} + \sum_{i\in S:(i,v_1)\in A} x_{i,v_1} + \sum_{(i,j)\in T(P)} x_{i,j} \leq |P|. \tag{34}$$

Proposition 5 *Given any elementary path $P = (v_1, \ldots, v_h)$, BTOUR inequality* (34) *is valid for TBF. Furthermore, it dominates inequality* (33).

Proof. If the third term in inequality (34) equals $|P|$, then the first term must be 0 (if 1, the last node of the path would be reached after its deadline) and the second must be 0 as well, as implied by the subtour. If instead $\sum_{(i,j)\in T(P)} x_{i,j} < |P|$, then the inequality (34) is implied by constraint (3) for node v_1, which clearly holds for TBF (see Proposition 1).

It is easy to see that inequality (34) dominates (33) as $L \subseteq \bigcup_{v\in V\setminus\{v_1\}} \{b \in B_{v_1} : (b', b) \in A_{\mathcal{B}} \text{ for some } b' \in L_v\}$ and hence $\sum_{v\in V\setminus S} \sum_{b\in L_v} y^b_{v,v_1} + \sum_{i\in S:(i,v_1)\in A} x_{i,v_1} \geq \sum_{v\in V\setminus\{v_1\}} \sum_{b\in L_v} y^b_{v,v_1} \geq \sum_{b\in L} z^b_{v_1}$. □

4 Separation Routines

In this section we describe the main steps of our separation routines for inequalities involving buckets presented in the previous section.

4.1 Separating Bucket SOP Inequalities

In their most general version, SOP inequalities (21), (22) and (25) cannot be separated in polynomial time. However, Ascheuer, Fischetti and Grötschel [7] developed an effective polynomial-time separation heuristic, based on theoretical results of Balas, Fischetti and Pulleyblank [9] on the Precedence Constrained TSP. Following the approach in [7], we develop a polynomial-time separation heuristic for bucket SOP inequalities. We next outline the main steps of our separation procedure for simple π_B-inequalities and simple (π_B, σ_B)-inequalities. Our separation procedures for the other bucket SOP inequalities are very similar.

Separating simple π_B-inequalities

Consider a set $S \subseteq V$ with $q \notin S$. Let $A^S = \{(b, b') \in A_{\mathcal{B}} : b, b' \notin \pi(B(S))\}$, i.e., it is the collection of arcs in the bucket graph with neither end preceding the buckets associated with the nodes in S. Given a solution (x^*, y^*, z^*) of the TBR, let $\tilde{x}$ be defined by

$$\tilde{x}_{ij} = \sum_{(i,j,b) \in \mu(A^S)} (y^b_{ij})^*.$$

Recall the definition of the simple π_B-inequality for the set S given in (28). The left hand side of this inequality evaluated at y^* satisfies

$$\sum_{(i,j,b) \in \mu(Y)} (y^b_{ij})^* = \sum_{(i,j) \in \delta(S)} \tilde{x}_{ij}.$$

Therefore the simple π_B-inequality above is violated if and only if $\tilde{x}$ violates the SEC (18). Given any $S' \supseteq S$ with $q \notin S'$, as $\pi_B(S) \subseteq \pi_B(S')$ and the left hand side of the simple π_B-inequality for the set S' is bounded above by $\sum_{(i,j) \in \delta(S')} \tilde{x}_{ij}$, it follows that the simple π_B-inequality for the set S' is violated if $\tilde{x}$ violates the corresponding SEC, but the converse is not true. Finally, note that as $\tilde{x} \leq x^*$, if x^* violates any SEC for some $S' \supseteq S$, then so does $\tilde{x}$. Thus, we can use flows in the graph $G = (V, A)$ with arc capacities given by $\tilde{x}$ to find some violated simple π_B-inequalities .

Our separation heuristic is as follows. For every $v \in V \setminus \{p, q\}$, let $S := \{v\}$ and execute the following steps:

1. Using the sets A^S, Y defined above, compute $\tilde{x}$.

2. Compute the max-flow z^0 from S to q in $G = (V, A)$ with the capacity of an arc (i, j) equal to $\tilde{x}_{ij}$. Let $\delta(S')$ be the corresponding min-cut. If $z^0 < 1$, then add the violated simple π_B-inequality for the set S' to the cut-pool.

3. If $\pi_B(S) = \pi_B(S')$, then terminate, otherwise set $S := S'$ and go to Step 1.

In Step 3, if $\pi_B(S) = \pi_B(S')$, then the weight of the cut $\delta(S')$ with arcs weights $\tilde{x}_{ij}$ equals the left hand side of the simple π_B-inequality based on S', but is an overestimate if $\pi_B(S) \neq \pi_B(S')$. Therefore, when $z^0 \geq 1$, the simple π_B-inequality based on S' is definitely not violated by y^* if $\pi_B(S) = \pi_B(S')$, but maybe violated if $\pi_B(S) \neq \pi_B(S')$. Recomputing $\tilde{x}$ after the assignment $S := S'$ yields a correct value of the left hand side of the simple π_B-inequality based on S'. The procedure described above acts as an exact separation algorithm for the subtour elimination constraints. Whenever this procedure fails to find a violated simple π_B-inequality starting from any node $v \in V \setminus \{p, q\}$, there are no violated SECs either. Note, however, that there may exist some violated bucket SECs. Our procedure for separating simple σ_B-inequalities is very similar.

Separating simple (π_B, σ_B)-inequalities

Following Ascheuer, Fischetti and Grötschel [7], we separate over simple (π_B, σ_B)-inequalities by considering sets X and Y of the form $X = \{u\}$, $Y = \{w\}$ with $u \prec w$ and $u, w \in V \setminus \{p, q\}$. For each $u, v, w \in V$ such that $u \prec v \prec w$, it can easily be shown that all the simple (π_B, σ_B)-inequalities arising from $X = \{u\}$, $Y = \{w\}$ are implied by those arising from $X = \{u\}$, $Y = \{v\}$ together with those arising from $X = \{v\}$, $Y = \{w\}$. Hence, when looking for violated simple (π_B, σ_B)-inequalities, we only consider pairs of nodes u, w such that $u \prec w$ and the precedence between u and w does not immediately follow from known precedences of the form $u \prec v \prec w$.

Given a pair of nodes u, w satisfying the above criteria, let $\tilde{W}$ and $\tilde{Q}$ be defined as in (29), with $X = \{u\}$, and $Y = \{w\}$. Let $A^{u,w} = \{(b, b') \in A_{\mathcal{B}} : b, b' \notin \tilde{W}, (b, b') \notin \tilde{Q}\}$. Given a solution (x^*, y^*, z^*) of the TBR, let $\tilde{x}$ be defined by

$$\tilde{x}_{ij} = \sum_{(i,j,b) \in \mu(A^{u,w})} (y^b_{ij})^*.$$

As in the case of simple π_B-inequalities, one can measure the left hand side of some simple (π_B, σ_B)-inequalities via flows in G. More precisely, for any set S with $u \in S$ and $w \in \bar{S}$, the left hand side of the simple (π_B, σ_B)-inequality equals the weight of the cut $\delta(S)$ in G, where the weight of an arc (i, j) is given by $\tilde{x}_{ij}$.

Our separation algorithm is as follows. For every pair of nodes u, w such that $u \prec w$ and it is not known that $u \prec v \prec w$ for some node v, we execute the following steps:

1. Using the set $A^{u,w}$ defined above, compute $\tilde{x}$.

2. Compute the max-flow z^0 from u to w in G (with arc capacities $\tilde{x}_{ij}$) and let $\delta(S)$ be the corresponding min-cut, with $u \in S$ and $w \in \bar{S}$. If $z^0 < 1$, then add the violated simple (π_B, σ_B)-inequality corresponding to S, u, w to the cut-pool.

As highlighted in the description of the separation procedures above, the *simple* bucket SOPs allow us to strengthen the SOPs by exploiting information from the y variables, even if all max-flow computations are performed in the space of x variables, and thus in much smaller graphs.

We implemented similar heuristics to separate non-simple bucket SOPs; these procedures explicitly work in the y-space and thus have to deal with larger graphs and are more time-consuming. Based on a limited amount of experimentation, we concluded that the additional improvements in the lower bound obtained by separating the more general inequalities was not worth the time spent in separation. Further, we often found a very large number of violated non-simple bucket SOPs; adding these made the subsequent LPs much harder to solve. Based on these considerations, we only work with the simple bucket SOPs in our final branch-and-cut code.

4.2 Separating Tournament and Bucket Tournament Constraints

To find violated tournament constraints we use an enumeration algorithm similar to the one described in [7]. Given a fractional solution x^*, the algorithm builds a tree for each node $v \in V \setminus \{p, q\}$ where every node of the tree associated with v corresponds to an elementary path ending at v. The tree for node v is built by starting with an empty path initially, and the existing paths are expanded by adding one more arc to the path at a time by following the arcs in the support of the fractional solution x^*. The paths are expanded in a depth-first fashion until a certain termination criterion is satisfied. Let $P = (u, \ldots, v)$ denote the path associated with a node of the enumeration tree. Exactly one of the following three cases holds true for P:

(i) The tournament inequality (31) associated with P is not violated. In this case we stop expanding the path P any further.

(ii) The tournament inequality (31) associated with P is violated and P is an infeasible path because it satisfies the simple infeasibility test (30). In this case we stop expanding P and add the violated tournament inequality associated with P to the cut-pool. (Before adding the inequality (31) to the cut-pool, we check if it can be strengthened to an inequality (32).)

(iii) The tournament inequality associated with P is violated but P does not satisfy the infeasibility test (30). In this case we generate a number of new paths by expanding P, one for each node $w \in V \setminus P$ such that $x^*_{wu} > 0$.

To find violated bucket tournament constraints (34), we modified the enumeration algorithm described above slightly by changing the third case to include a check for bucket tournament constraints. Before we expand the path, we simply check if the bucket tournament constraint (34) associated with P is violated. If it is, we add the corresponding inequality to the cut-pool.

Note that the procedure above stops extending a path P as soon as P satisfies inequality (31) or P is discovered to be infeasible according to the trivial condition (30). If the first condition happens, clearly any path obtained by expanding P also satisfies inequality (31) as well as inequality (34). Further, if the x solution is integer and does not contain any cycle, then the trivial condition (30) is sufficient at providing an exact separation for tournament constraints (31). Indeed, given a Hamiltonian path P from p to q, P is feasible if and only if all the subpaths contained in P are feasible according to the trivial condition (30). In [7], the authors state that this algorithm runs in

polynomial time because for any fractional solution there can only be a polynomial number of paths that violate the tournament inequality. The proof is attributed to Savelsbergh, but, unfortunately, there is no written proof of this claim. In any case, in our computational experiments we observed that the algorithm terminates after a small number of iterations.

5 Solution Approach

In this section we describe the main components of our solution approach. Given an instance of the problem, the first step is to produce an associated Time Bucket Formulation. To this end, we preprocess the input data and then divide the time windows into buckets using an LP-based iterative heuristic. The B&C algorithm starts with an initial formulation based on these buckets and strengthens it at every node with valid inequalities. We also apply a fast primal heuristic at every node and try to generate a feasible tour, i.e., an upper bound for the problem. We describe these ideas in detail below. Throughout we assume that the triangle inequality holds for the traveling times, that is, for all $(i, j) \in A$, the shortest path distance from node i to node j is equal to θ_{ij}.

5.1 Preprocessing

Given an instance of the problem, we first modify the input graph $G = (V, A)$ and the time windows associated with nodes in V to obtain an equivalent instance with fewer arcs and tighter time windows. We also build a list of node precedences.

5.1.1 Tightening the Time Windows

We tighten time windows following the procedure described in [7]. We cycle through the following four steps until no more changes can be made to the time windows. Recall that R_k and D_k denote the earliest and latest start time at node $k \in V$, respectively, and $V^+(k)$ and $V^-(k)$ denote the set of nodes that are connected to k by arcs from k and arcs to k, respectively.

1. $R_k \leftarrow \max\{R_k, \min_{i \in V^-(k)}\{R_i + \theta_{ik}\}\}$ $\quad \forall\ k \in V$ s.t. $V^-(k) \neq \emptyset$.
2. $R_k \leftarrow \max\{R_k, \min\{D_k, \min_{j \in V^+(k)}\{R_j - \theta_{kj}\}\}\}$ $\quad \forall\ k \in V$ s.t. $V^+(k) \neq \emptyset$.
3. $D_k \leftarrow \min\{D_k, \max\{R_k, \max_{i \in V^-(k)}\{D_i + \theta_{ik}\}\}\}$ $\quad \forall\ k \in V$ s.t. $V^-(k) \neq \emptyset$.
4. $D_k \leftarrow \min\{D_k, \max_{j \in V^+(k)}\{D_j - \theta_{kj}\}\}$ $\quad \forall\ k \in V$ s.t. $V^+(k) \neq \emptyset$.

5.1.2 Building the Node Precedence List

After tightening the time windows, we build a list of pairwise precedence relationships between the nodes. We again follow the ideas described in [7]. These precedence relationships are used in the preprocessing phase as well as in separating SOP inequalities. Recall that if a node $i \in V$ has to be visited before $j \in V$ in any feasible tour, then we say $i \prec j$. First, we set $p \prec j$ for all nodes $j \in V \setminus \{p\}$ and $i \prec q$ for all $i \in V \setminus \{q\}$. Then, we repeat the following two steps until no new precedence relationships are found. We emphasize that the second step below is not implied by the first one.

1. For $i, j \in V \setminus \{p, q\}$, if $R_j + \theta_{ji} > D_i$, then set $i \prec j$.

2. For $i, j, k \in V \setminus \{p, q\}$ if $i \prec j$ and $j \prec k$, then set $i \prec k$.

5.1.3 Deleting Arcs

After tightening the time windows and building a precedence list, we delete some of the arcs in A using logical implications. First we delete all $(i, j) \in A$ such that $j \prec i$ and all $(i, j) \in A$ such that $i \prec k \prec j$ for some $k \in V \setminus \{i, j\}$. Next, we consider all remaining $(i, j) \in A$ and for each $k \in V \setminus \{i, j\}$ we check if the condition

$$(k \prec i \text{ or } k \prec j \text{ or } R_i + \theta_{ij} + \theta_{jk} > D_k) \text{ and } (i \prec k \text{ or } j \prec k \text{ or } R_k + \theta_{ki} + \theta_{ij} > D_j)$$

holds. In such a case, we conclude that arc (i, j) cannot be part of a feasible tour and therefore we delete it from A.

5.2 Building the Bucket Graph

There are many possible ways to divide the time windows into buckets and how the buckets are chosen affects the quality of the formulation significantly. When the total number of buckets increases, the size of the formulation increases and typically this leads to a continuous relaxation which is harder to solve, but at the same time, tighter. Note that if the total number of buckets is fixed, one has to decide how to distribute these buckets among individual nodes. If the number of buckets allocated to a node is given, one still has to decide how to split the associated time window into buckets.

In our initial experiments, we tried two natural ways to obtain buckets. The first one was to split each time window into a fixed number of buckets and make the buckets of a node (approximately) the same size. The second one was to fix the width of a bucket, and then divide all time windows into buckets accordingly. In this approach, nodes with wider time windows are divided into more buckets.

Subsequently, we observed that it is useful to treat nodes non-uniformly in dividing their windows into buckets, and to use non-uniform buckets for a single node. We therefore developed a heuristic that builds the bucket graph iteratively by solving LP relaxations of the TBR and refining the buckets by inspecting the corresponding LP solutions. We describe this approach below.

5.2.1 Building an Initial Set of Buckets

To build the initial set of time buckets, we first identify time instants for each node when a feasible tour cannot visit the node. In other words, for any node $i \in V$ we identify $t \in W_i$ such that the start time of node i cannot be t. Though identifying all such time instants is clearly a hard problem, a simple yet sufficient test for $t \in W_i$ is to see if $t - \theta_{ki} \in W_k$ for some $k \in V$ such that $(k, i) \in A$. If there is no such $k \in V$, and if $t \neq R_i$, then a feasible tour cannot have start time t at node i and we call t a *hole* in the time window W_i (see also Section 2.3). After identifying holes

in the time windows we compute the bucket set B_i of node $i \in V$ by combining consecutive time instants that are not interrupted by holes.

5.2.2 Bucket Refinement Heuristic

Given a collection of buckets, we first preprocess these buckets and then construct the associated bucket graph following the steps described later in detail in Section 5.3. Using this bucket graph $G' = (\mathcal{B}, A_{\mathcal{B}})$, we solve the associated linear program TBR-LP and obtain the optimal solution $(\bar{x}, \bar{z}, \bar{y})$ (such a notation for the optimal solution of the continuous relaxation is used instead of the more usual (x^*, z^*, y^*) to avoid confusion in the following). Let $\bar{B} = \{b \in \mathcal{B} : \bar{z}_i^b > 0 \text{ where } b \text{ is a bucket of node } i\}$ denote the collection of buckets in this solution that have non-zero activity. The bucket refinement heuristic divides each bucket $b \in \bar{B}$ with $r_b \neq d_b$ into two new buckets $b^1 = [r_b, \tau - 1]$ and $b^2 = [\tau, d_b]$ of possibly different size. We pick the break point τ to minimize what we call the "negative waiting time" associated with the current solution at bucket b. More precisely, we choose τ as follows:

First we decompose $\bar{z}_i^b$ into $\bar{z}_i^t$ for $t \in [r_b, d_b]$ as follows:

$$\bar{z}_i^t = \sum_{(k,b') \in B^-(t)} \bar{y}_{ki}^{b'}, \qquad t = r_b + 1, \ldots, d_b$$

where $B^-(t) = \{(k, b') : b' \in B_k,\ (b', b) \in A_{\mathcal{B}},\ r_{b'} + \theta_{ki} = t\}$. For $t = r_b$, we set $\bar{z}_i^t = \bar{z}_i^b - \sum_{t'=r_b+1}^{d_b} z_i^{t'}$. We say that the negative waiting time at bucket b is

$$\sum_{t=r_b}^{d_b} (t - r_b) \bar{z}_i^t.$$

We then choose the break point τ to minimize the sum of the negative waiting times in the new buckets $b^1 = [r_b, \tau - 1]$ and $b^2 = [\tau, d_b]$. In other words, $\tau \in [r_b + 1, d_b]$ is chosen to be the index that minimizes

$$\alpha(b, \tau) = \sum_{t=r_b}^{\tau-1} (t - r_b) \bar{z}_i^t + \sum_{t=\tau}^{d_b} (t - \tau) \bar{z}_i^t.$$

5.2.3 Iterative Bucket Refinement Heuristic

The iterative bucket refinement heuristic starts with the initial set of buckets described in Section 5.2.1. These buckets are then refined by applying the bucket refinement heuristic described in Section 5.2.2 repeatedly for k iterations. At the end of the k-th iteration, the optimal LP value LB_0 of the initial formulation is compared to the optimal LP value LB_1 of the final formulation. If $\lceil LB_1 \rceil = \lceil LB_0 \rceil$, then the refinement process is terminated. If, on the other hand, $\lceil LB_1 \rceil > \lceil LB_0 \rceil$, then the current buckets are considered as the initial set of buckets and the whole refinement procedure is repeated for another k iterations. This heuristic clearly depends on the number of refinement iterations performed before the termination condition is checked. After preliminary testing, we chose $k = 5$ for all our computational experiments. The formulation produced by this iterative heuristic is the one we use in the branch-and-cut algorithm.

5.3 Bucket preprocessing

Given an instance of the problem and a collection of buckets $\mathcal{B} = \bigcup_{i \in V} B_i$, we perform the following three preprocessing steps: First we refine $\mathcal{B}$ by subdividing some of the buckets $b \in \mathcal{B}$ until the resulting collection of buckets $\mathcal{B}'$ satisfies what we call the *bucket triangle inequality*. We then build a list of bucket precedences. Finally, using the precedences, we delete some arcs of the bucket graph $G' = (\mathcal{B}, A_{\mathcal{B}'})$. We describe these steps below.

5.3.1 Imposing bucket triangle inequality

Consider the bucket graph $G' = (\mathcal{B}, A_{\mathcal{B}})$, let $i, j, k \in V$ be three distinct nodes of the original graph G and let $b \in B_i$. Furthermore assume that bucket b can reach $b_1 \in B_k$ directly and $b_2 \in B_k$ via a bucket of j. More precisely, this means that $(b, b_1) \in A_{\mathcal{B}}$ and $(b, b'), (b', b_2) \in A_{\mathcal{B}}$ for some $b' \in B_j$. Buckets b_1 and b_2 do not have to be distinct. We say that the bucket triangle inequality is violated if bucket $b_2 \in B_k$ is strictly earlier than $b_1 \in B_k$, that is, if $r_{b_2} < r_{b_1}$. This can happen when there is a large "negative waiting time" $r_b + \theta_{ij} - r_{b'}$ at the intermediate node j.

Whenever we identify a violated bucket triangle inequality, we divide the intermediate bucket b' that causes the violation into two new buckets so that this particular violation does not happen with the new buckets. An easy way to do this is to break the bucket $[r_{b'}, d_{b'}]$ into two new buckets $[r_{b'}, r_b + \theta_{ij} - 1]$ and $[r_b + \theta_{ij}, d_{b'}]$.

Even though the validity of the TBF does not require elimination of these bucket triangle inequality violations, we observed that the LP relaxation of the TBF becomes noticeably weaker when such violations are present. We therefore always check for these violations and refine the buckets until the bucket graph is free of them. We also note that separating bucket SOP inequalities is easier when the bucket graph does not have any bucket triangle inequality violations.

5.3.2 Building the Bucket Precedence List

We build a list of precedence relationships among nodes and buckets by extending some of the ideas discussed in Section 5.1.2. These precedence relationships are used to reduce the size of the bucket graph as well as in separating bucket SOP inequalities. Recall that if any feasible tour that visits node $i \in V$ at bucket $b \in B_i$ has to visit node $j \in V$ after node i, we say $b \prec j$. Similarly, if any feasible tour that visits node $j \in V$ at bucket $b \in B_j$ has to visit node $i \in V$ before node j, we say $i \prec b$. First, for every $i \prec j$, we set $b \prec j$ for all $b \in B_i$ and $i \prec b$ for all $b \in B_j$. We then iterate the following steps until no new precedence relationships are found. Let $i, j, k \in V$ and $b \in B_i$.

1. If $r_b + \theta_{ij} > D_j$, set $j \prec b$. If $R_j + \theta_{ji} > d_b$, set $b \prec j$.
2. If $b \prec j$ and $j \prec k$, set $b \prec k$. If $j \prec k$ and $k \prec b$, set $j \prec b$.

Note that Step 1 above is valid provided that the bucket triangle inequality holds.

5.3.3 Deleting Arcs of the Bucket Graph

We delete some of the arcs of the bucket graph G' using logical implications. First we inspect the bucket precedence list and if $b \prec j$ for some $b \in B_i$ and $j \in V$, we delete the bucket-arcs $(b', b) \in A_{\mathcal{B}}$ for all $b' \in B_j$. Similarly, we delete the bucket-arcs $(b, b') \in A_{\mathcal{B}}$ for all $b' \in B_j$ if $j \prec b$.

Next, we consider all remaining $(b, b') \in A_{\mathcal{B}}$. Let $b \in B_i$ and $b' \in B_j$ for $i, j \in V$. If there exists a node $k \in V \setminus \{i, j\}$ such that $b \prec k$ and $k \prec b'$ or such that

$$(k \prec b \text{ or } k \prec b' \text{ or } r_b + \theta_{ij} + \theta_{jk} > D_k) \text{ and } (b \prec k \text{ or } b' \prec k \text{ or } R_k + \theta_{ki} + \theta_{ij} > D_j),$$

we conclude that bucket-arc (b, b') cannot be part of a feasible tour and we delete it from $A_{\mathcal{B}}$.

5.4 Cutting

We add cuts at every node of the B&C tree. We separate violated inequalities in the following order:

1. Simple π_B-inequalities (26),
2. Simple σ_B-inequalities (27),
3. Simple (π_B, σ_B)-inequalities (29),
4. Tournament constraints (31) and Bucket Tournament constraints (34) at the same time.

We do not use the classical TSPTW cutting planes such as SOP inequalities or the subtour elimination constraints as they are dominated by simple bucket SOP inequalities. Furthermore, we restrict ourselves to the simple bucket SOP inequalities instead of using the more general versions due to practical reasons. After some experimentation, we observed that one can use a small number of simple bucket SOP inequalities to achieve essentially the same bound as one can by using a large number of general bucket SOP inequalities, but with noticeably less separation time.

5.5 Primal Heuristic

At each node of the branch-and-cut tree we try to construct a feasible tour using the current LP solution. The heuristic starts with the node $p \in V$ and extends the tour one node at a time by choosing the next node among a list of candidate nodes that look promising according to the LP solution. A node j is considered promising if the reduced cost associated with the variable x_{ij} is zero. More precisely, let P denote the partial path constructed so far, i be the last node on this path and t_i be the start time at node i. In addition, let $\hat{c}_{ij}$ denote the reduced cost of variable x_{ij} in the current solution. We first construct a set of good candidate nodes to extend the path

$$N = \{j \in V \setminus (\{q\} \cup P) : \hat{c}_{ij} = 0, t_i + \theta_{ij} \leq D_j\}.$$

If $N = \emptyset$, we cannot extend the current path and we stop. Otherwise, we choose $j = \text{argmin}_{h \in N}\{D_h - \theta_{ih}\}$ to be the next node, i.e., according to a chronological rule, and augment the current path accordingly. The heuristic terminates with a feasible tour when all nodes except the last node q are visited by the current path.

6 Computational Results

We solve instances of the TSPTW by solving the corresponding Time Bucket Formulation with a branch-and-cut algorithm implemented in `C++` using calls to the Cplex 10.0 [19] callable library. All our computational results are obtained on a workstation with a Intel(R) 2.40 GHz processor running the SUSE Linux 10.1 Operating System. We use the Cplex default branching strategy, and disable all Cplex cuts. At every node of the branch-and-cut tree, we invoke our separation routine to find violated globally valid cuts, and also run our primal heuristic to search for a feasible solution to the TSPTW. For each instance, we impose a limit of 5 hours on the computing time. In the following two sections we present our computational experiments on asymmetric and symmetric instances separately.

6.1 Asymmetric Instances

Our first test set consists of the asymmetric TSPTW instances introduced by Ascheuer [5]. These instances are derived from a practical scheduling application, and have between 12 and 233 nodes. An extended computational study on these instances was carried out in Ascheuer, Fischetti and Grötschel [7] using a branch-and-cut approach. Their paper is our main reference in this section. Based on the results in [7], we divide the 50 asymmetric instances into 32 "easy" problems that are solved to optimality within 5 hours of CPU time in [7], and 18 "hard" problems. To the best of our knowledge, the only "hard" instance solved in a subsequent work is `rbg042a` (see Focacci, Lodi and Milano [18]).

In Tables 1 to 4, we report on the performance of our TBF-based branch-and-cut algorithm on the above asymmetric instances. The column headings common to many of these tables are explained below.

Column headings	**Description**
Prob.	name of the problem instance
$\|V\|$	number of nodes
$\|A\|$	number of arcs (after preprocessing)
lpLB	lower bound at the root node given by the LP relaxation
rLB	lower bound at the root node after cuts
gLB	lower bound at termination (at the time limit)
bestUB	best upper bound known in prior literature (in bold if optimal).
UB	best upper bound found (in bold if optimal)
#cuts	number of generated cuts
#nodes	number of B&C nodes
%gap	percentage of the integrality gap closed
CPU	total CPU time spent in seconds
CPU-sep	CPU time spent in separation routines

In Table 1, we compare our TBF-based method with the branch-and-cut method proposed in [7] (often denoted as AFG) on the 32 easy instances. For each instance, and for both methods, we report the percentage value of the lower bound at the root node (after the addition of cutting planes and before branching) with respect to the optimal solution OPT (computed as 100 * rLB / OPT), the number of branch-and-cut nodes enumerated, and the overall CPU time in seconds.

Table 1: Time Bucket Formulation vs. Ascheuer et al. [7]: comparison on the "easy" instances.

			Ascheuer et al. [7] (AFG)			Time Bucket Formulation (TBF)		
Prob.	$\|V\|$	$\|A\|$	%rLB	#nodes	CPU	%rLB	#nodes	CPU
rbg010a	12	54	99.3	2	0.1	100.0	1	0.0
rbg016a	18	79	98.9	2	0.2	100.0	0	0.0
rbg016b	18	167	93.7	76	8.8	97.2	2	0.2
rbg017.2	17	200	100.0	0	0.0	100.0	0	0.0
rbg017a	19	176	100.0	0	0.1	100.0	0	0.0
rbg017	17	122	100.0	4	0.8	99.3	0	0.0
rbg019a	21	71	100.0	0	0.0	100.0	0	0.0
rbg019b	21	211	98.9	820	54.6	99.5	1	0.3
rbg019c	21	229	95.8	58	8.7	96.8	42	0.9
rbg019d	21	156	99.7	2	0.8	100.0	6	0.2
rbg020a	22	95	100.0	0	0.2	100.0	0	0.0
rbg021.2	21	237	100.0	0	0.2	100.0	0	0.1
rbg021.3	21	256	97.8	340	27.2	98.4	62	2.7
rbg021.4	21	264	98.9	72	5.8	100.0	1	0.2
rbg021.5	21	268	98.8	76	6.6	100.0	1	0.3
rbg021.6	21	358	99.3	2	1.4	100.0	1	0.3
rbg021.7	21	375	96.2	24	4.3	100.0	0	0.6
rbg021.8	21	380	97.7	254	17.4	98.5	10	1.4
rbg021.9	21	380	97.0	320	26.1	98.5	23	2.8
rbg021	21	229	95.8	58	8.8	96.8	42	0.9
rbg027a	29	479	99.3	6	2.3	99.3	2	1.4
rbg031a	33	388	100.0	0	1.7	100.0	0	0.2
rbg033a	35	421	100.0	0	1.9	99.8	6	0.9
rbg034a	36	535	99.5	2	1.0	100.0	6	1.9
rbg035a.2	37	940	95.2	96	64.8	100.0	3	5.3
rbg035a	37	477	100.0	0	1.8	100.0	2	0.2
rbg038a	40	486	100.0	13204	4232.2	100.0	9	1.5
rbg040a	42	539	92.0	1756	751.8	96.6	25	3.6
rbg050a	52	1629	100.0	6	18.6	100.0	2	24.5
rbg055a	57	765	99.9	2	6.4	100.0	19	3.5
rbg067a	69	843	99.9	2	6.0	100.0	23	3.6
rbg125a	127	1824	99.5	56	229.8	100.0	8	9.6
avg.			98.5	538.8	171.6	99.4	9.3	2.1

These instances seem to be easy for both AFG and TBF. In particular, they both obtain a strong lower bound at the root node. However, it is clear that our method explores substantially fewer branch-and-cut nodes. One cannot really compare the reported times of the methods as they are obtained on different machines[1].

[1]The results reported in [7] were obtained on a SUN SPARC Station 10 by using the branch-and-cut framework

In Table 2 we present our results on the 18 "hard" instances, and also compare them with AFG. For each problem, the table reports its size and the best upper bound known in the literature (and the optimal value for `rbg042a`) when it is different from the value obtained by AFG; these bounds are obtained in [18]. Then, for both AFG and TBF we report the lower bound at the root node, the lower and upper bounds at the end of the computation, and the number of branch-and-cut nodes enumerated. For TBF, we also report the initial LP relaxation value, the total CPU time and the time spent in separation routines. Finally, for TBF, in the last column (denoted as r%gap) we give the improvement at the root over the AFG root bound, measured as a fraction of the gap between the AFG root bound and the best known upper bound BUB (either from the literature, including AFG, or computed by TBF), i.e., 100*(TBF rLB - AFG rLB)/(BUB - AFG rLB).

Table 2: Time Bucket Formulation vs. Ascheuer et al. [7]: comparison on the "hard" instances.

				Ascheuer et al. [7] (AFG)				Time Bucket Formulation (TBF)							
Prob.	$\|V\|$	$\|A\|$	bestUB	rLB	gLB	UB	#nodes	lpLB	rLB	gLB	UB	#nodes	CPU	CPU-sep	r%gap
rbg041a	43	628	403	361	382	417	23396	389.4	392	402	**402**	270	146.8	2.7	75.61
rbg042a	44	762	**411**	394	409	435	22300	391.9	404	411	**411**	161	188.3	2.4	58.82
rbg048a	50	1288	492	454	455	527	25222	484.8	487	487	**487**	0	129.2	0.1	100.00
rbg049a	51	1083	488	408	418	501	17486	460.9	464	474	486	991	> 18000	30.0	71.79
rbg050b	52	1175	527	447	453	542	8600	482.6	493	495	—	382	> 18000	20.9	57.50
rbg050c	52	1396	—	507	509	536	25184	512.7	517	517	—	289	> 18000	16.8	34.48
rbg086a	88	926	—	1042	1049	1052	12208	1035.6	1049	1051	**1051**	16	4.9	0.6	77.78
rbg092a	94	1367	1109	1084	1102	1111	8828	1075.1	1091	1093	**1093**	40	90.4	5.4	77.78
rbg132.2	132	3126	—	1053	1069	1125	4336	1068.8	1081	1083	**1083**	25	2761.1	10.6	93.33
rbg132	132	1575	—	1323	1348	1400	7628	1324.3	1353	1360	**1360**	64	37.6	3.5	81.08
rbg152.3	152	6191	—	1521	1525	1594	2558	1517.1	1538	1539	**1539**	20	10353.3	27.4	94.44
rbg152	152	2125	—	1759	1770	1792	5038	1753.3	1780	1783	**1783**	29	43.7	3.0	87.50
rbg172a	174	2837	—	1777	1787	1897	3434	1739.9	1796	1799	**1799**	22	425.5	8.4	86.36
rbg193.2	193	6031	—	1969	1981	2093	1726	1986.0	2010	2013	—	5	> 18000	22.3	33.06
rbg193	193	3050	—	2386	2388	2452	2790	2394.3	2414	2414	**2414**	10	159.6	3.4	100.00
rbg201a	203	3287	—	2158	2159	2296	3282	2134.6	2187	2189	**2189**	20	462.7	9.4	93.55
rbg233.2	233	7588	—	2146	2152	2304	1200	2170.9	2181	2184	—	22	> 18000	26.9	22.15
rbg233	233	3766	—	2635	2647	2786	1106	2676.7	2689	2689	**2689**	15	749.4	8.2	100.00
avg.							9795.7					132.3	5864.0	11.2	74.74

Table 2 shows that the branch-and-cut algorithm based on the Time Bucket Formulation solves 13 out of the 18 problems, sometimes in just a few minutes, and improves upon the lower bound obtained by AFG in the remaining 5 problems[2]. We also improve upon the best known upper bound for one of the five problems we are unable to solve, namely `rbg049a`. Moreover, the root lower bound with TBF is generally much stronger, thus dramatically reducing the number of B&C nodes required to solve the problem. Indeed, for 9 out of the 18 problems, the lower bound we

ABACUS [1].

[2]A couple of AFG results, namely on instances `rbg092a` and `rbg125a`, have some numerical issues or typos in [7]. More precisely, (i) we obtained a feasible (and optimal) solution of value 1093 for instance `rbg092a` while [7] reports a final lower bound value of 1102 and (ii) we obtained a feasible (and optimal) solution of value 1409 for instance `rbg125a` while [7] reports an optimal value of 1410. The solutions of these instances – as well as of the others – are available upon request to the authors.

obtain by solving the LP relaxation (obtained after a few rounds of bucket refinement and before adding cuts) is better than the final lower bound obtained by AFG, often after the exploration of many thousands of nodes in the branch-and-cut tree. However, in some cases the strength of the LP relaxation of the TBF comes at the cost of difficult-to-solve LPs, due to the large number of variables. Consequently, for some instances we can explore very few branch-and-cut nodes within the time limit, especially for `rbg193.2` and `rbg233.2`. On the negative side, on 4 of the 5 instances we are unable to solve to optimality, our algorithm does not find any feasible solution within the time limit. In all these problems the number of nodes explored is rather small because of the size and difficulty of the LP relaxations. In addition, the primal heuristic we are using is not particularly sophisticated.

By imposing as a cut off value that of the best known upper bound plus one, our branch-and-cut algorithm is able to solve to optimality instance `rbg049a` (value 484), and to find for instances `rbg050b` and `rbg050c` tighter lower and upper bounds, [502,516] and [520,527], respectively, within an extended time limit of 10 hours.

6.1.1 Impact of Cutting Planes on TBF-based Branch-and-Cut

In this section we discuss the impact of cuts on our branch-and-cut algorithm.

The aim of Table 3 is to analyze the effectiveness of the bucket inequalities. We consider the 18 hard instances, perform iterative bucket refinement to obtain the desired time bucket relaxation, and then invoke our separation routines till we cannot find violated cuts. Here we do not perform branching. Table 3 compares the lower bound obtained using the classical cuts which do not exploit the buckets in any way versus the bound obtained using our default setting. In both cases, we measure the integrality gap closed, i.e., the fraction of the gap between the LP relaxation value and the best known upper bound which is closed using cuts from each category. While separating the classical cuts alone, we generate, in order, subtour elimination constraints, SOPs (21), (22), (25) and finally TOURs (31).

It is clear that the bucket inequalities are useful: the "node inequalities" close 57.06 % of the integrality gap on the average, whereas we can close 63.57 % of the integrality gap with bucket inequalities. In some cases this improvement is crucial for the overall performance of the algorithm. For example, instance `rbg132.2` is solved in 7564.8 CPU seconds and 132 nodes with node cuts instead of 2761.1 CPU seconds and 25 nodes with bucket cuts, and instance `rbg152.3` cannot be solved to optimality within the time limit (while it was solved in 10353.3 CPU seconds, see Table 2).

Although the computing time required by the code using bucket cuts is on average almost double that when using node cuts alone, note that (i) the separation times for the bucket version as reported in Table 2 are not too large and (ii) most of the difference in the averages of Table 3 is due to instances `rbg152.3` and `rbg193.2`. For these instances, bucket cuts are considerably more time-consuming than node cuts, mainly during LP reoptimization.

To complete the analysis of the impact of cutting planes on branch-and-cut performance, we

Table 3: Node cuts vs. bucket cuts at the root node: comparison on the "hard" instances.

	TBF + Node cuts			TBF + Bucket cuts		
Prob.	#cuts	%gap	CPU	#cuts	%gap	CPU
rbg041a	12	2.29	2.5	37	13.75	3.5
rbg042a	21	49.54	4.7	80	59.74	6.5
rbg048a	12	85.58	112.6	49	99.24	141.6
rbg049a	12	4.79	82.7	57	9.32	94.7
rbg050b	24	17.66	121.1	127	21.74	175.2
rbg050c	13	7.32	337.2	48	14.17	385.5
rbg086a	37	74.75	0.8	81	83.80	1.2
rbg092a	55	74.71	3.9	137	84.13	5.6
rbg132.2	45	67.04	128.7	191	80.09	314.1
rbg132	47	67.18	5.9	72	78.21	8.6
rbg152.3	31	92.03	4461.1	197	92.99	8671.2
rbg152	38	77.90	12.1	61	88.68	26.0
rbg172a	58	90.80	35.8	176	94.49	56.1
rbg193.2	52	21.27	1820.7	209	22.08	5981.1
rbg193	49	100.00	178.3	49	100.00	103.7
rbg201a	60	91.34	74.6	156	94.97	76.8
rbg233.2	44	6.95	4398.9	130	6.95	4877.1
rbg233	78	95.95	257.4	110	100.00	336.0
avg.	38.2	57.06	668.8	109.3	63.57	1181.4

experimented by adding to the version of the code separating only node cuts (as in Table 3 above) one bucket cut type at the time, namely (26), or (27), or (29), or (34). Though all four versions are more effective than the basic one with only node cuts, the version which adds (π_B, σ_B)-inequalities (29) is the most interesting one. Indeed, this version closes 62.49 % of the integrality gap on the "hard" instances, generating 174.3 cuts on average. However, the additional 1 % gap closed using all other bucket cuts is important for the overall performance of the algorithm. Further, the version with all bucket cuts is less time-consuming, as fewer cuts are added on the average, namely 109.3 cuts versus 174.3 cuts.

6.1.2 Experimenting with Different Time Discretizations

We performed two sets of experiments to evaluate the impact of the time discretization. In Table 4 we compare the Time Indexed Formulation presented in Section 2.2 with the Time Bucket Formulation based on our heuristic partitioning of time windows into buckets.

The table compares the two formulations in terms of (i) size (number of buckets $|\mathcal{B}|$ and number of arcs $|A_{\mathcal{B}}|$), (ii) lower bound given by the LP relaxation as a percentage of the best known solution (%lpLB), and (iii) total CPU time spent solving the initial LP (CPU-LP). In addition, for TBF the table also reports the lower bound at the root node (after cuts) again as a percentage of the best known solution (%rLB) and total CPU time spent at the root node (CPU-root).

Table 4 clearly shows that the quality of the bound of TIF is only slightly stronger than that of TBF at the price of being two orders of magnitude more time-consuming. In addition, once cuts are added to TBF the root lower bound is stronger than that from TIF, yet takes much less time

Table 4: TIF vs. TBF, root node on the "hard" instances.

	TIF				TBF					
Prob.	$\|\mathcal{B}\|$	$\|A_{\mathcal{B}}\|$	%lpLB	CPU-LP	$\|\mathcal{B}\|$	$\|A_{\mathcal{B}}\|$	%lpLB	CPU-LP	%rLB	CPU-root
rbg041a	24,675	247,377	96.52	659.2	1,638	15,996	97.01	2.3	97.51	3.5
rbg042a	25,750	314,857	95.86	1016.2	1,379	18,687	95.38	2.8	98.30	6.5
rbg048a	44,931	914,756	99.79	18723.6	3,635	77,508	99.59	86.1	100.00	141.6
rbg049a	39,937	679,686	95.06	13020.2	3,896	70,998	94.86	65.8	95.47	94.7
rbg050b	41,387	751,953	91.84	11943.3	3,794	75,414	91.65	99.5	93.55	175.2
rbg050c	53,518	1,115,444	96.08	31925.0	4,643	111,482	95.71	283.6	96.46	385.5
rbg086a	43,871	338,813	98.57	756.3	1,011	8,935	98.57	0.5	99.81	1.2
rbg092a	52,755	530,504	98.54	3913.7	1,439	16,609	98.44	1.7	99.82	5.6
rbg132.2	155,993	2,546,567	98.80	98155.6	5,838	104,898	98.71	75.8	99.82	314.1
rbg132	76,951	639,210	97.72	2572.0	2,781	25,214	97.43	3.6	99.49	8.6
rbg152.3	269,654	7,540,079	—	—	11,794	350,735	98.64	1106.9	99.94	8671.2
rbg152	89,942	842,037	98.37	5103.2	3,760	37,732	98.37	7.8	99.83	26.0
rbg172a	104,905	1,122,761	96.83	14959.5	4,968	57,586	96.72	18.5	99.83	56.1
rbg193.2	229,153	4,804,593	—	—	8,503	195,995	94.89	304.2	96.03	5981.1
rbg193	115,993	1,231,735	99.21	10274.7	7,179	87,328	99.21	46.5	100.00	103.7
rbg201a	121,713	1,292,053	97.53	19903.6	6,776	77,318	97.53	38.0	99.91	76.8
rbg233.2	277,398	6,019,216	—	—	11,681	273,930	94.23	421.8	94.66	4877.1
rbg233	140,623	1,515,251	99.59	23620.7	9,037	108,668	99.55	82.2	100.00	336.0
avg. all	106063.8	1,802,605.1	97.36	17103.1	5208.4	95,279.6	97.03	147.1	98.36	1181.4
avg.	75529.6	938,866.9	97.36	17103.1	4118.3	59,624.9	97.25	54.3	98.65	115.7

to obtain.

It is interesting that for `rbg041a` the lower bound of TIF is weaker than that of TBF. After spending some time looking for a bug in our code or an error in our reasoning (it is very intuitive that TIF should be provably tighter), we realized that TBF plus bucket preprocessing can indeed be tighter than TIF with bucket preprocessing. We present a small example demonstrating this in the Appendix.

As a final experiment, we compared our iterative bucket refinement procedure (IBR for short, see Section 5.2.3) with two straightforward strategies. For a given instance of the problem, let N indicate the overall number of buckets obtained with IBR. It is also possible to partition the time windows into approximately N buckets using one of the following procedures.

- Uniform buckets on nodes (UBN). Time window of node $i \in V$ is divided into $\min\{|W_i|, \lceil N/|V| \rceil\}$ buckets where $|W_i| = D_i - R_i + 1$ is the size of the time window. Time buckets associated with the same node are constructed to have (almost) the same size.

- Uniform buckets on times (UBT). Let $T = \sum_{i \in V} |W_i|$ be the overall number of time instants. The Time window of node $i \in V$ is divided into $\lceil N * (|W_i|/T) \rceil$ buckets. In this case, all time buckets are constructed to have approximately the same size.

These two alternative procedures by construction give formulations that are very similar in size to the one given by IBR. However, the LP bound given by the IBR is significantly better than both of them. In particular, the improvement on the quality of the LP bound (with no cuts) of IBR

with respect to UBN is 23.14 % and with respect to UBT is 21.86 %.

6.2 Symmetric Instances

The symmetric TSPTW instances we consider are derived from the `rc` instances proposed by Solomon [28] in the context of the Vehicle Routing Problem with Time Windows (VRPTW). Our TSPTW instances are obtained from the single-vehicle decomposition of the VRPTW solutions in the literature (i.e., Rochat and Taillard [26] and Taillard et al. [29]). The 27 TSPTW instances have up to 44 cities and have been studied, among others, by Pesant et al. [24] and by Focacci, Lodi and Milano [18]. For the underlying VRPTW instances, the travel costs or times between cities are computed as the Euclidean distances between their coordinates. Pesant et al. [24] showed that an optimal solution computed with travel costs truncated to two decimal places may even become infeasible with respect to the travel costs truncated to four decimal places. To construct symmetric instances for our branch-and-cut code we followed the recipe by Focacci, Lodi and Milano [18]: we create integer travel times by multiplying the Euclidean distances by 10,000 and then rounding each scaled distance to the nearest integer value. Since the travel time associated with each arc (i, j) is computed as $\theta_{ij} = c_{ij} + p_i$, where p_i is the processing time associated with node i, we applied the same scaling operation to the processing times and time windows.

In Table 5 we present the results obtained by running our TBF-based branch-and-cut code on the above symmetric instances treated as asymmetric ones by replacing each edge between two nodes by two arcs in opposite directions. The reference results for the symmetric TSPTW are those in [18] where 23 of the 27 instances were solved to optimality within 1,800 CPU seconds on a Pentium III 700 MHz PC.

The table reports the usual pieces of information already encountered in the previous tables and compares the best solutions found in [18] (or, in the case of instances `rc203.0`, `rc203.1`, `rc204.2` and `rc208.0`, the best solutions reported in [24]) with the TBF-based branch-and-cut algorithm. Our code is able to solve all the 23 instances solved in [18] plus 2 more among the 4 unsolved instances[3].

The only remaining unsolved instances, namely `rc204.2` and `rc208.0` are the largest instances both in terms of nodes (40 and 44, respectively) and arcs (1530 and 1964, respectively). For these instances, as in the case of difficult asymmetric ones, our algorithm cannot find a feasible solution within the time limit. However, using a cutoff value equal to the best upper bound in [18] plus one, the branch-and-cut code solved instance `rc204.2` to optimality within the time limit (value 378.40) and improved significantly the lower bound (value 364.21) on instance `rc208.0`.

[3] A couple of results in [18], namely on instances `rc203.2` and `rc205.2`, show a rounding error confirmed by the authors [22]. The optimal solutions for these instances have value 337.47 and 434.70, respectively.

Table 5: TBF branch-and-cut on symmetric benchmark instances.

				Time Bucket Formulation (TBF)										
Prob.	$\|V\|$	$\|A\|$	UB [18]	$\|\mathcal{B}\|$	$\|A_{\mathcal{B}}\|$	UB	lpLB	rLB	gLB	%rLB	#cuts	#nodes	CPU-root	CPU
rc201.0	25	169	**378.62**	116	588	**378.62**	371.24	378.62	378.62	100.00	4	0	0.0	0.0
rc201.1	28	157	**374.70**	107	485	**374.70**	340.32	374.70	374.70	100.00	4	0	0.0	0.0
rc201.2	28	209	**427.65**	140	748	**427.65**	400.50	427.65	427.65	100.00	8	0	0.0	0.0
rc201.3	19	127	**232.54**	70	324	**232.54**	216.81	229.51	232.54	98.70	24	2	0.0	0.1
rc202.0	25	440	**246.22**	581	6420	**246.22**	235.67	246.22	246.22	100.00	10	0	0.2	0.2
rc202.1	22	298	**206.53**	188	1620	**206.53**	186.35	206.53	206.53	100.00	33	0	0.1	0.1
rc202.2	27	379	**341.77**	234	1510	**341.77**	310.44	341.77	341.77	100.00	14	0	0.1	0.1
rc202.3	26	433	**367.85**	475	4045	**367.85**	331.74	367.85	367.85	100.00	39	0	0.2	0.2
rc203.0	35	1084	384.8	1510	30311	**377.45**	303.29	360.11	377.45	95.41	934	1906	23.1	3437.7
rc203.1	37	1144	357.3	2656	41241	**356.99**	262.55	348.44	356.99	97.60	387	353	36.8	2722.7
rc203.2	28	548	**337.46**	508	5927	**337.47**	289.17	337.47	337.47	100.00	54	0	0.8	0.9
rc204.0	32	1003	**221.45**	322	8023	**221.45**	184.82	221.35	221.45	99.95	123	0	1.3	2.8
rc204.1	28	798	**205.37**	620	14500	**205.37**	180.88	203.31	205.37	99.00	87	40	4.8	14.9
rc204.2	40	1530	379.0	4303	97694	—	315.98	365.51	366.02	96.44	1080	1081	271.6	> 18000
rc205.0	26	375	**251.65**	327	2659	**251.65**	223.50	251.65	251.65	100.00	16	0	0.1	0.1
rc205.1	22	229	**271.22**	202	1275	**271.22**	254.08	271.22	271.22	100.00	7	0	0.0	0.0
rc205.2	28	347	**434.69**	372	2311	**434.70**	355.74	434.70	434.70	100.00	89	0	0.2	0.3
rc205.3	24	280	**361.24**	207	951	**361.24**	336.48	361.24	361.24	100.00	11	0	0.0	0.0
rc206.0	35	675	**485.23**	513	7311	**485.23**	374.73	473.18	485.23	97.52	317	225	1.6	73.3
rc206.1	33	686	**334.73**	686	10988	**334.73**	291.35	328.59	334.73	98.17	216	105	4.4	85.0
rc206.2	32	663	**335.37**	687	10689	**335.37**	276.34	331.34	335.37	98.80	130	27	3.5	84.8
rc207.0	37	1030	**436.69**	1091	20068	**436.69**	358.15	434.50	436.69	99.50	129	4	7.8	19.0
rc207.1	33	860	**396.36**	903	16253	**396.36**	312.52	389.95	396.36	98.38	146	72	8.7	34.9
rc207.2	30	771	**246.41**	540	9536	**246.41**	219.92	239.40	246.41	97.16	210	143	1.1	116.4
rc208.0	44	1964	381.1	3174	111867	—	279.00	355.57	359.46	93.30	703	455	556.2	> 18000
rc208.1	27	755	**239.04**	492	12741	**239.04**	186.06	229.71	239.04	96.10	589	2220	1.3	990.6
rc208.2	29	869	**213.92**	1252	30775	**213.92**	184.39	213.92	213.92	100.00	28	0	7.9	7.9
avg.										98.74	199.7	245.7	34.5	1614.5

7 Conclusions

In this paper we presented an extended formulation for the Traveling Salesman Problem with Time Windows, a well known generalization of the classical TSP where each node must be visited within a given time window. In particular, we proposed a quite general idea based on partitioning time windows into sub-windows. We showed how to implement this idea to obtain a strong formulation, and also how to generate strong valid inequalities and incorporate them in a classical branch-and-cut framework.

We tested the overall branch-and-cut algorithm on hard benchmark instances from the literature, arising from a practical scheduling application in the asymmetric case, and randomly generated in the symmetric one. The results show that the proposed formulation is effective in practice for tackling the TSPTW. We solved several previously unsolved benchmark instances.

Future directions of work include, on the methodological side, the extension and generalization of the presented approach to other contexts in which the time window component is relevant. On the computational side, some difficult instances remained unsolved. There are at least three

ingredients of the current framework that could be improved. Namely,

1. Off line experiments obtained imposing a cut off value to the branch-and-cut algorithm suggested that more sophisticated primal heuristics could lead to improved results on both asymmetric and symmetric instances.

2. Table 3 has shown that some of the instances for which bucket inequalities are not much better than node inequalities are among the ones we cannot solve. A more clever exploitation of the bucket information might lead to better cutting planes and, possibly, to optimal solutions of these problems.

3. There is a trade-off between very detailed partitions of the time windows into sub-windows (leading to very strong bounds) and the difficulty of the resulting LPs. It may be possible to reduce the time to solve these LPs by working with a subset of variables and pricing the remaining ones only when necessary in a column generation framework.

Appendix

Consider the following "toy" TSPTW instance with $V = \{1, 2, \ldots, 7\}$ and with time windows $W_1 = [0, 0]$, $W_2 = [1, 2]$, $W_3 = [1, 2]$, $W_4 = [5, 12]$, $W_5 = [5, 9]$, $W_6 = [5, 11]$, $W_7 = [20, 20]$. Assume the graph $G = (V, A)$ to be complete. Traveling times are $\theta_{ij} = 1$ for $(i, j) \in A_1^\theta = \{(1, 2), (1, 3), (2, 3), (2, 6), (3, 2), (5, 4)\}$, $\theta_{ij} = 3$ for $(i, j) \in A_3^\theta = \{(2, 4), (3, 5), (4, 6)\}$, $\theta_{ij} = 4$ for $(i, j) = (3, 4)$ and $\theta_{ij} = 2$ for all the other arcs. Traveling costs are $c_{ij} = 0$ for $(i, j) = (4, 6)$, $c_{ij} = 1$ for $(i, j) \in A_1^c = \{(1, 2), (2, 3), (3, 4), (3, 5), (4, 5), (4, 7), (5, 6), (5, 7), (6, 5), (6, 4), (6, 7)\}$ and $c_{ij} = 2$ for all the other arcs. The triangle inequality holds for the traveling times θ_{ij} but not for the costs c_{ij} because of the arc $(4, 6)$ with cost zero.

The optimal tour is $(1, 2, 3, 4, 5, 6, 7)$ where the nodes are visited at times $t_1 = 0$, $t_2 = 1$, $t_3 = 2$, $t_4 = 6$, $t_5 = 8$, $t_6 = 10$, $t_7 = 20$. The cost of the optimal tour is $c_{opt} = 6$. LP-TIF yields a lower bound $c^*_{TIF} = 5.5$ that corresponds to the fractional solution $x_{12} = x_{23} = 1$, $x_{34} = x_{35} = x_{46} = x_{47} = x_{56} = x_{57} = x_{64} = x_{65} = 0.5$. Note in particular that node 3 is visited at bucket $[2, 2] \in B_3$ with $z_3^{[2,2]} = 1$, arc $(3, 4)$ is used with $x_{34} = 0.5$ and hence node 4 is visited at bucket $[6, 6] \in B_4$ with $z_4^{[6,6]} = 0.5$. Thus, the arc $(i, j) = (4, 6)$ with cost 0 can be used with $x_{46} = 0.5$ because the bucket-arc $(b, b') \in A_{\mathcal{B}}$, $b = [6, 6] \in B_4$, $b' = [9, 9] \in B_6$ is used with $y_{46}^{[6,6]} = 0.5$.

Now consider a time bucket relaxation where nodes 2 and 3 have only one bucket, i.e. $B_2 = \{[1, 2]\}$ and $B_3 = \{[1, 2]\}$, while the time windows of all the other nodes are completely discretized as in TIF. With this choice of the buckets, LP-TBR yields the same lower bound as TIF, $c^*_{TBR} = 5.5$, that corresponds exactly to the same fractional x solution. In this case, node 3 is visited at bucket $[1, 2] \in B_3$ with $z_3^{[1,2]} = 1$, arc $(3, 4)$ is used with $x_{34} = 0.5$ and hence node 4 is visited at bucket $[5, 5] \in B_4$ (instead of [6,6]) with $z_4^{[5,5]} = 0.5$, because of the negative waiting time introduced by the bucket $[1, 2] \in B_3$. Thus, the arc $(i, j) = (4, 6)$ with cost 0 is again used with $x_{46} = 0.5$, because the bucket-arc $(b, b') \in A_{\mathcal{B}}$, $b = [5, 5] \in B_4$, $b' = [8, 8] \in B_6$ can be used with $y_{46}^{[5,5]} = 0.5$.

However, if we apply the so-called bucket preprocessing to the above TIF and TBR, then the bucket-arc $(b, b') \in A_{\mathcal{B}}$, $b = [5, 5] \in B_4$, $b' = [8, 8] \in B_6$ is removed, since for node $k = 5$ we have $b \prec k$ and $r_b + \theta_{46} + \theta_{65} > D_k$. Thus, the lower bound yielded by LP-TBR improves to $c^*_{TBR} = 6$, that corresponds to the optimal tour $(1, 2, 3, 4, 5, 6, 7)$, while the optimal solution of LP-TIF does not change, since the bucket-arc $(b, b') \in A_{\mathcal{B}}$, $b = [6, 6] \in B_4$, $b' = [9, 9] \in B_6$ is not removed by the bucket preprocessing.

Acknowledgments

Part of this research was carried out when the fourth author was a summer intern at the Department of Mathematical Sciences of the IBM T.J. Watson Research Center, whose support is strongly acknowledged.

References

[1] ABACUS, A Branch-And-CUt System, `http://www.informatik.uni-koeln.de/abacus/`.

[2] J. Albiach, J. M. Sanchis, and D. Soler. An asymmetric TSP with time windows and with time-dependent travel times and costs: An exact solution through a graph transformation. *European Journal of Operational Research* **189**, 789–802, 2008.

[3] L. Appelgren. A column generation approach for a ship scheduling problem. *Transportation Science* **3**, 53–68, 1969.

[4] L. Appelgren. Integer programming methods for a vessel scheduling problem. *Transportation Science* **5**, 62–74, 1971.

[5] N. Ascheuer. *Hamiltonian path problems in the on-line optimization of flexible manufacturing systems.* PhD thesis, Technische Universität Berlin, Berlin, Germany, 1995.

[6] N. Ascheuer, M. Fischetti, M. Grötschel. A polyhedral study of the asymmetric travelling salesman problem with time windows. *Networks* **36**, 69–79, 2000.

[7] N. Ascheuer, M. Fischetti, M. Grötschel. Solving the asymmetric travelling salesman problem with time windows by branch-and-cut, *Mathematical Programming, Ser. A* **90**, 475–506, 2001.

[8] E.K. Baker. An exact algorithm for the time-constrained travelling salesman problem. *Operations Research* **31**, 938–945, 1983.

[9] E. Balas, M. Fischetti and W.R. Pulleyblank. The precedence-constrained asymmetric traveling salesman polytope. *Mathematical Programming, Ser. A* **68**, 241–265, 1995.

[10] E. Balas, N. Simonetti. Linear time dynamic programming algorithms for new classes of restricted TSPs: a computational study. *INFORMS Journal on Computing* **13**, 56-75, 2001.

[11] Y. Caseau, P. Koppstein. A rule-based approach to a time-constrained traveling salesman problem. *Proceedings of the 2nd International Symposium of Artificial Intelligence and Mathematics*, Fort Lauderdale, FL. 1992.

[12] N. Christofides, A. Mingozzi, P. Toth. State space relaxation procedures for the computation of bounds to routing problems. *Networks* **11**, 145–164, 1981.

[13] Concorde TSP Solver. `http://www.tsp.gatech.edu/concorde.html`

[14] W. Cook. Personal communication, 2008.

[15] J.-F. Cordeau, G. Desaulniers, J. Desrosiers, M.M. Solomon, F. Soumis. VRP with Time Windows. In P. Toth, D. Vigo, eds. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications. pp. 157–194, 2002.

[16] J. Desrosiers, Y. Dumas, M.M. Solomon, F. Soumis. Time constrained routing and scheduling. In M.O. Ball, T.L. Magnanti, C.L. Monma, G.L. Nemhauser, eds. *Network Routing*. Elsevier, Amsterdam, The Netherlands. pp. 35–139, 1995.

[17] Y. Dumas, J. Desrosiers, E. Gelinas, M.M. Solomon. An optimal algorithm for the travelling salesman problem with time windows. *Operations Research* **43**, 367–371, 1995.

[18] F. Focacci, A. Lodi, M. Milano. A hybrid exact algorithm for the TSPTW. *INFORMS Journal on Computing* **14**, 403–417, 2002.

[19] ILOG Cplex, `http://www.ilog.com/products/cplex`.

[20] A. Langevin, M. Desrochers, J. Desrosiers, F. Soumis. A two-commodity flow formulation for the traveling salesman and makespan problem with time windows. *Networks* **23**, 631–640, 1993.

[21] A. Levin. Scheduling and fleet routing models for transportation systems. *Transportation Science* **5**, 232–255, 1971.

[22] A. Lodi. Personal communication, 2009.

[23] A. Mingozzi, L. Bianco, S. Ricciardelli. Dynamic programming strategies for the travelling salesman problem with time windows and precedence constraints. *Operations Research* **45**, 365–377, 1997.

[24] G. Pesant, M. Gendreau, J.-Y. Potvin, J.M. Rousseau. An exact constraint logic programming algorithm for the travelling salesman problem with time windows. *Transportation Science* **32**, 12–29, 1998.

[25] G. Pesant, M. Gendreau, J.-Y. Potvin, J.M. Rousseau. On the flexibility of constraint programming models: from single to multiple time windows for the travelling salesman problem. *European Journal of Operational Research* **117**, 253–263, 1999.

[26] Y. Rochat, E. D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics* **1**, 147–167, 1995.

[27] M.W.P. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research* **4**, 285–305, 1985.

[28] M. M. Solomon. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research* **35**, 254–265, 1987.

[29] E. D. Taillard, P. Badeau, M. Gendreau, F. Guertin, J.-Y. Potvin. A new neighborhood structure for the vehicle routing problems with time windows. Publication CRT-95-66, Centre de Recherche sur les Transports, Université de Montréal, Montréal, Quebec, Canada, 1995.

[30] A. Tramontani. *Enhanced mixed integer programming techniques and routing problems.* PhD thesis, DEIS, University of Bologna, Bologna, Italy, 2009.

[31] X. Wang, A. C. Regan. Local truckload pickup and delivery with hard time window constraints. *Transportation Research Part B* **36**, 97–112, 2002

[32] X. Wang, A. C. Regan. On the convergence of a new time window discretization method for the traveling salesman problem with time window constraints. *Computers & Industrial Engineering* **56**, 161–164, 2009.